目錄

Linux 101 Hacks

Copyright & Disclaimer

Introduction

作者：**Ramesh Natarajan**

网址：*www.thegeekstuff.com*

译者：**WrFly**

联系方式：*mr.wrfly@gmail.com*

本书简介：

- Linux进阶技巧
- 巧妙的命令组合
- Bash某些技巧
- 一共一百零一个(包括充数的)
- 最后有个奖励章(额外技巧)

鉴于本书的版权为2009-2011，所以我并不侵权是吧：)

除去第九章和第十章, 本书所有命令均在 **Ubuntu** 下操作, 如果您使用的是别的发行版, 那么可能会有些许出入.

另外, 作者的有些命令是在充数... 所以我里面有些许吐槽... 见谅...

再另, 我并不是按照原作原原本本的翻译的, 里面穿插着某些我的看法, 建议英文好的阅读原文.

WrFly 2016年1月

第一章 关于 cd

```
➤ man cd
No manual entry for cd
```

这么常用的cd竟然没有man！是因为太简单了么？

如果你认为它太简单而轻视它，那就是你的不对了，不信？下面的六种技巧就会让你大吃一惊！

用 CDPATH 来重新定义目录

如果你经常 `/etc` 下的目录，每次都要输入 `/etc/xxx` 真的有点烦，但是当你设定 `CDPATH` 后，你就可以直接在 `CDPATH` 目录和当前目录下游走了～

比如：

```
➤ pwd
/home/mr
➤ cd nginx
bash: cd: nginx: No such file or directory #当然进不去这个目录
➤ echo $CDPATH #默认的CDPATH是空的

➤ CDPATH=/etc #把CDPATH设为/etc，就可以进入/etc下的目录了
➤ cd nginx
/etc/nginx
➤ pwd
/etc/nginx #进去了～
➤
```

如果你想长期使用这个方法，比如，经常游走于 `/etc` , `/var` , `/etc/nginx` ,那么就可以把 `CDPATH` export一下，像这样：

`export CDPATH=.:~:/etc:/var:/etc/nginx`

把这句话写入 `.bashrc` 就可以永久生效了。

CD的别名

这也是一个小技巧，用途很广泛，用 `alias` 设置别名：

比如，如果要想返回上一层目录，你可能会这样做：`cd ..` ，是不是很烦？

那么这种方法可能会让你有点吃惊：

```
➤ pwd
/home/mr
➤ cd .. #返回上一层目录
➤ pwd
/home
➤ cd /home/mr
➤ alias ..='cd ..' #利用alias
➤ ..
➤ pwd #同样返回了上一层目录
/home
```

然后你可以把好多有趣的命令都用 `alias` 缩短，我在github上整理了一下我常用的 `alias` 命令，感兴趣的话可以去看看 :P

Here: https://github.com/wrfly/bash_aliases

与CD有关的Functions()

下面说一种与 `alias` 差不多的 `function` ：

```
➤ mkdir -p /tmp/subdir1/subdir2/subdir3
mkdir: created directory ‘/tmp/subdir1’
mkdir: created directory ‘/tmp/subdir1/subdir2’
mkdir: created directory ‘/tmp/subdir1/subdir2/subdir3’
➤ cd /tmp/subdir1/subdir2/subdir3
➤ pwd
/tmp/subdir1/subdir2/subdir3
➤
```

是不是又有点烦？

下面就让 `function` 来解救你吧～

```
function mcd () {
    mkdir -p "$@" && eval cd "\"\$$#\"";
}
```

然后，瞧好了：

```
➤ mcd /tmp/1/2/3/4/5/6/7
mkdir: created directory ‘/tmp/1’
mkdir: created directory ‘/tmp/1/2’
mkdir: created directory ‘/tmp/1/2/3’
mkdir: created directory ‘/tmp/1/2/3/4’
mkdir: created directory ‘/tmp/1/2/3/4/5’
mkdir: created directory ‘/tmp/1/2/3/4/5/6’
mkdir: created directory ‘/tmp/1/2/3/4/5/6/7’
➤ pwd
/tmp/1/2/3/4/5/6/7
➤
```

为什么会这样呢？这是因为我们在当前环境下新建了一个 `function` ，这个 `function` 的功能就是新建目录，然后进入我们新建的目录。

当然，这应该是最简单的 `function` 了吧，把一堆常用的 `function` 写入你的 `.bashrc` 里面，会让你很舒服的。

同样，我的一些常用 `function` 也在 `github` 上面，跟 `alias` 是放一块儿的哦。

后退后退！

接着上面的目录，我们目前在这里：

```
➤ pwd
/tmp/1/2/3/4/5/6/7
➤
```

然后我去 `/tmp` 下写了 `wrfly到此一游` 之后，又想返回刚才的目录了，该怎么办？

Terminal里可没有后退键给我按！

```
➤ pwd
/tmp/1/2/3/4/5/6/7
➤ cd /tmp
➤ echo ""wrfly到此一游""
wrfly到此一游
➤ echo "wrfly到此一游" > ttttest
➤ pwd
/tmp
➤ cd - ##看清了吗？我可用了两个井号键呢！
/tmp/1/2/3/4/5/6/7
➤ pwd
/tmp/1/2/3/4/5/6/7
➤
```

的确，Terminal不给我们后退键，因为里面就没有键可以按啊，哈哈哈哈，不过嘛，这么多命令总有一个可以达到我们的目的，就比如刚才的 **`cd -`** ，通过这个命令我们就后退到了之前的目录了。

其实这里面还有一些道道，比如：

```
➤ pwd
/tmp/1/2/3/4/5/6/7
➤ echo $OLDPWD
/tmp
➤ cd -
/tmp
➤ echo $OLDPWD
/tmp/1/2/3/4/5/6/7
➤
```

这个 `$OLDPWD` 就是上一层目录的意思，当然还有 `$PWD` 这个变量，表面上看来跟 `pwd` 是一样的（因为 `pwd` 还有一个 `-P` 的参数可以用，可以显示 `soft link` 的真实路径，所以他们并不是完全相同）

再插句题外话，说一下 `pwd` ：

```
➤ ln -s /tmp/1/2/3/4/5/6/7 7
➤ ll 7
lrwxrwxrwx 1 mr mr 18 12月 23 15:51 7 -> /tmp/1/2/3/4/5/6/7/
#为什么是18这么大呢？因为'/tmp/1/2/3/4/5/6/7'一共有18个字节啦
➤ cd 7
➤ pwd
/tmp/7
➤ pwd
/tmp/7
➤ pwd -P
/tmp/1/2/3/4/5/6/7
➤ echo $PWD
/tmp/7
➤
```

是不是很好玩呢？

操纵目录栈

刚才我还特地请教了下我们宿舍打ACM的杰神，讲了一下堆、栈、队列之间的差别和联系，然后，这一小节的题目还算不错，栈，先进后出。

- `dirs` 显示当前目录栈的内容
- `pushd` 入目录栈
- `popd` 出目录栈

那么这三个命令都有什么用处呢？

不罗嗦，看操作：

```
➤ for i in {1..4};do mkdir dir_$i;done
mkdir: created directory ‘dir_1’
mkdir: created directory ‘dir_2’
mkdir: created directory ‘dir_3’
mkdir: created directory ‘dir_4’
➤ cd dir_1
➤ pushd . #pushd完成之后总会进入这一个目录
/tmp/dir_1 /tmp/dir_1
➤ cd /tmp/dir_2
➤ pushd . #而且还会显示当前的目录栈内容
/tmp/dir_2 /tmp/dir_2 /tmp/dir_1
➤ cd /tmp/dir_3
➤ pushd .
/tmp/dir_3 /tmp/dir_3 /tmp/dir_2 /tmp/dir_1
➤ cd /tmp/dir_4
➤ pushd .
/tmp/dir_4 /tmp/dir_4 /tmp/dir_3 /tmp/dir_2 /tmp/dir_1
➤ dirs #第一个是dir_4，因为栈的顶总是当前目录
/tmp/dir_4 /tmp/dir_4 /tmp/dir_3 /tmp/dir_2 /tmp/dir_1
```

那么到现在为止我们的目录栈看起来是这样的：

```
1 - /tmp/dir_1
2 - /tmp/dir_2
3 - /tmp/dir_3
4 - /tmp/dir_4
```

接下来就轮到 `popd` 啦：

```
➤ pwd
/tmp/dir_4
➤ dirs #dirs显示目录栈内容
/tmp/dir_4 /tmp/dir_4 /tmp/dir_3 /tmp/dir_2 /tmp/dir_1
➤ popd #每次目录出栈之后都会显示目录栈的内容
/tmp/dir_4 /tmp/dir_3 /tmp/dir_2 /tmp/dir_1
➤ pwd #并且进入那个目录
/tmp/dir_4
➤ popd
/tmp/dir_3 /tmp/dir_2 /tmp/dir_1
➤ pwd
/tmp/dir_3
➤ popd
/tmp/dir_2 /tmp/dir_1
➤ pwd
/tmp/dir_2
➤ popd
/tmp/dir_1
➤ pwd
/tmp/dir_1
➤ popd #栈空了，没有了
bash: popd: directory stack empty
➤
```

看了这么多，这些组合到底有什么用？其实吧，如果你手速够快记忆力够好的话，这些对你没啥用，不过呢，像我这样比较懒的人，这东西就有用了，比如，我要在几个目录之间切换，有不想每次都输入那些命令，怎么办？就用这种方法，因为pushd不仅进入了这个目录，而且还给了我一个后退的途径，比 `cd -` 更高级一点。甚至我可以用 `function cd() { pushd $@ &> /dev/null; }` 来将 `pushd` 变成 `cd` ,反正我也不亏，是不是 :P

自动更正目录名

其实感觉这个功能不是很有用，因为大多数情况下我们都是用 `tab` 直接补全目录名的，不过作者既然在这里说(chong)了(shu)，那本着尊重作者的原则我也应该啰嗦下：

`shopt -s cdspell`

就是这货，开启更正目录开关。

比如：

```
➤ cd /tmp/dir_1
➤ ls
hello
➤ cd hhllo
bash: cd: hhllo: No such file or directory
➤
```

没错，进不去。

但是当你开启那个神奇的开关之后：

`shopt -s cdspell`

```
➤ shopt -s cdspell
➤ cd hhllo
hello
➤ pwd
/tmp/dir_1/hello
➤
```

进去了～这就是自动更正目录名的效果。

其实不仅如此，`shopt` 还有好多可以玩的地方，有兴趣的同学可以探索下哦～

`shopt -s xxxx` 是开启xxxx

`shopt -u xxxx` 是关闭xxxx

```
➤ shopt
autocd                  off
cdable_vars             off
cdspell                 on
checkhash               off
checkjobs               off
checkwinsize            on
cmdhist                 on
compat31                off
compat32                off
compat40                off
compat41                off
compat42                off
complete_fullquote      on
direxpand               off
dirspell                off
dotglob                 off
execfail                off
expand_aliases          on
extdebug                off
extglob                 on
extquote                on
failglob                off
force_fignore           on
globstar                off
globasciiranges         off
gnu_errfmt              off
histappend              on
histreedit              off
histverify              off
hostcomplete            off
huponexit               off
interactive_comments    on
lastpipe                off
lithist                 off
login_shell             off
mailwarn                off
no_empty_cmd_completion off
nocaseglob              off
nocasematch             off
```

```
nullglob             off
progcomp             on
promptvars           on
restricted_shell      off
shift_verbose        off
sourcepath           on
xpg_echo             off
```

第二章 - 基本命令

基本命令包含了大部分系统操作常用的命令，这本书一共写了20个。

我在翻译的时候会尽量按照作者的本意，不过偶尔也会加点作料，让它们更有味道～

毕竟这只是个开篇，所以也没必要啰嗦。

各位看官瞧好吧 :P

Grep 命令

先说句题外话免得我最后忘了：`ag`，貌似比`grep`更强大。

> grep - print lines matching a pattern

通俗的说，就是查找。

怎么查找呢？看语法说明:

`grep [options] pattern [files]`

比如在`/etc/passwd`里面查找 **root**

```
➤ grep root /etc/passwd
root:x:0:0:root:/root:/bin/bash
➤
```

> 查找 **bash**

```
➤ grep bash /etc/passwd
root:x:0:0:root:/root:/bin/bash
mr:x:1000:1000:mr,,,:/home/mr:/bin/bash
```

> **-v** 反向查找，即查找不带pattern的内容

如果还是用文件的话，输出的东西就多了，所以不在此演示。

> **-c** 只是看下有多少行匹配到了

```
➤ grep -cv ':' /etc/passwd
0
➤ grep -c bash /etc/passwd
2
➤
```

更多选项

> **-i** 忽略大小写 忽略pattern的大小写
>
> **-r** 递归搜索 在某个目录下搜索全部匹配的文件
>
> **-l** 只显示文件名 不显示匹配到的行
>
> **-E** 扩展模式 后面可接正则表达式,更强大

扩展阅读

- 15个实用的Grep命令
- 强大的 'Z' 命令系列(Zcat, Zgrep, Zless, Zdiff)
- Grep组合表达式

Grep与正则表达式

其实学过正则表达式之后这些就不算啥事儿了,建议先学一下正则然后再来看.

不过直接看下面的更功利一些,毕竟看完了就可以直接用了.

下面这个是测试文的内容:

```
➤ cat -n fortune
     1	All generalizations are false, including this one.
     2			-- Mark Twain
     3	Many pages make a thick book, except for pocket Bibles which are on very
     4	very thin paper.
     5
     6	Remark of Dr. Baldwin's concerning upstarts: We don't care to eat toadstools
     7	that think they are truffles.
     8			-- Mark Twain, "Pudd'nhead Wilson's Calendar"
     9
    10	Sheriff Chameleotoptor sighed with an air of weary sadness, and then
    11	turned to Doppelgutt and said 'The Senator must really have been on a
    12	bender this time -- he left a party in Cleveland, Ohio, at 11:30 last
    13	night, and they found his car this morning in the smokestack of a British
    14	aircraft carrier in the Formosa Straits.'
    15			-- Grand Panjandrum's Special Award, 1985 Bulwer-Lytton
```

匹配句首 '^'

```
➤ grep -ni '^A' fortune
1:All generalizations are false, including this one.
14:aircraft carrier in the Formosa Straits.'
```

> **-n** 的意思是显示行号

匹配句尾 '$'

```
➤ grep -ni '\.$' fortune
1:All generalizations are false, including this one.
4:very thin paper.
7:that think they are truffles.
```

这里的 **'\.$'** 是以 **.** 字符结尾的意思, 中间的反斜杠是逃逸字符,起转义的效果

匹配空行 '^$'

```
➤ grep '^$' fortune -n
5:
9:
```

任意单个字符 '.'

```
➤ grep -n '.re' fortune
1:All generalizations are false, including this one.
3:Many pages make a thick book, except for pocket Bibles which a
re on very
6:Remark of Dr. Baldwin's concerning upstarts: We don't care to
eat toadstools
7:that think they are truffles.
11:turned to Doppelgutt and said 'The Senator must really have b
een on a
```

> **\b** 的意思是单词分界线,也许是空格,也许是tab

```
➤ grep -n '\b.re\b' fortune
1:All generalizations are false, including this one.
3:Many pages make a thick book, except for pocket Bibles which a
re on very
7:that think they are truffles.
```

零次或多次重复 '*'

```
➤ grep -n 'l*' fortune
1:All generalizations are false, including this one.
2:        -- Mark Twain
3:Many pages make a thick book, except for pocket Bibles which a
re on very
4:very thin paper.
5:
6:Remark of Dr. Baldwin's concerning upstarts: We don't care to
eat toadstools
7:that think they are truffles.
8:        -- Mark Twain, "Pudd'nhead Wilson's Calendar"
9:
10:Sheriff Chameleotoptor sighed with an air of weary sadness, a
nd then
11:turned to Doppelgutt and said 'The Senator must really have b
een on a
12:bender this time -- he left a party in Cleveland, Ohio, at 11
:30 last
13:night, and they found his car this morning in the smokestack
of a British
14:aircraft carrier in the Formosa Straits.'
15:        -- Grand Panjandrum's Special Award, 1985 Bulwer-Lytt
on
```

显示的不是很好,因为没有高亮什么的.

其实这一部分完全可以当做正则来讲, 毕竟正则才是核心, grep只是一个工具, 怎么用,还是看你的能力如何了.

曾经看过一个30分钟入门正则表达式, 个人感觉挺不错的, 因为我也是从那里入的门, 所以也推荐看这一部分的人也去搜下那个, 作者写的够简单易懂了.

这一段写的有点长, 但是, 当你学会正则的时候你就会发现, 这东西真棒! (这是我BB的, 跟原作者无关哈...)

扩展阅读

- 正则与Grep (part I)
- 正则与Grep (part II)

Find 命令

Find命令是一个常用的搜索命令,可以找到任何你想找到的文件.

基本语法: `find [pathnames] [conditions]`

按照文件名查找

```
➤ find . -name '*py*' #支持正则表达式
./hello-world.py
./49.pyc
./helloword.py
./helloword.pyc
./49-2.py
./49.py
```

按文件大小查找

```
➤ find . -type f -size +200 #默认是KB,也可以改成100M或者1G
./kis.tar.bz2
./kismet/Kismet-20151220-13-28-40-1.pcapdump
./kismet/Kismet-20151220-13-28-40-1.netxml
./kismet/Kismet-20151220-13-28-40-1.nettxt
./kismet.tar.gz
./system.map
```

> -type 指定文件类型

type	文件类型
f	普通文件
d	目录文件
b	随机存储的设备文件，如硬盘，光盘等存储设备
c	持续输入的设备文件，如鼠标，键盘
p	有名管道
l	链接文件(link)
s	socket文件

按修改时间查找

查找60天之前修改过的文件

```
➤ find . -mtime +60
./test.php
➤ ls -l test.php
-rw-rw-r-- 1 mr mr 0 12月 12  2001 test.php #我之前用touch修改过它
```

查找到了文件就删除

```
find / -type f -name *.tar.gz -size +100M -exec rm -f {} \;
```

来我们详细说下这条命令，首先是一个 `-type` ，指定了文件类型，然后跟着一个 `-name` ，告诉find要找什么文件，再接着是一个size,规定了要找的文件的大小，最后的 `-exec` 是执行某项命令，命令是 `rm -rf` ，参数是 `{}` ，这个 `{}` 是啥意思? 它代表的就是find查找到的文件，最后的 `\;` 是告诉find，这条语句已经结束了,后面的事儿就不用它管了.

其实上面的语句完全可以用另一个参数来代替， `-delete` 参数.

完整的命令如下: `find / -type f -name *.tar.gz -size +100M -delete`

扩展阅读

- Find 基础命令 I

- Find 进阶命令 II

重定向

其实重定向这里没什么好说的, 不过为了照顾新手以及满足原作者凑数的目的,还是在这里扯一扯.

如果你看过鸟哥的私房菜,你就不要再继续看下去了.

第一个要说的是标准输出重定向 **>**

```
➤ cat helloword.py
#!/usr/bin/python

print "Hello world!"

➤
```

```
➤ cat helloword.py > /dev/null
➤
```

第二个例子把输出的信息都重定向到 **/dev/null/** 了

> /dev/null 可以理解为无底洞, 所有进入里面的数据流都会消失~

上面例子中的箭头就是重定向符号, 还有其他的重定向符号,比如 **2>**, **&>**, 你就会问了,有没有 **1>** 呢? 当然有啊, **1>** 就是 **>**, 只是把 1 省略了而已.

如果要了解重定向的相关知识, 还请搜索 "linux 重定向".

实在没啥可讲的...

Join命令

这个命令挺有用的, 用来把两个文件合并到一起.

> join - join lines of two files on a common field

比如, 下面有两个文件:

```
➤ cat join_1
1 a
2 b
3 c
4 d
5 e
➤ cat join_2
1 A
2 B
3 C
4 D
5 E
➤
```

然后用 `join` 把他们合起来:

```
➤ join join_1 join_2
1 a A
2 b B
3 c C
4 d D
5 e E
```

有什么特殊要求嘛? 当然有, 要求就是两个文件有相同的字段.

tr 命令

`tr` 的全称是 translate, 翻译, 或者转换, 随你怎么理解.

下面看一下它的功能:

```
➤ cat join_1
1 a
2 b
3 c
4 d
5 e
➤ cat join_1 | tr a-z A-Z
1 A
2 B
3 C
4 D
5 E
➤ tr a-z A-Z < join_1
1 A
2 B
3 C
4 D
5 E
➤
```

看到了吗? `tr` 就是转换字符的, 不仅可以转换大小写哦, 还能删除, 替换等一些操作.

比如:

```
➤ echo hello world | tr -d world
he
➤
➤ echo hello world | tr -s e a
hallo world
➤
```

不过删除替换都比较鸡肋, 不如 `sed` 好用. 毕竟术业有专攻.

Xargs命令

Xargs是用来做什么的呢? OK, 先看下他能干什么:

```
➤ ls
1  2  3  4
➤ ls | xargs ls -l
-rw-rw-r-- 1 mr mr 0 12月 26 20:46 1
-rw-rw-r-- 1 mr mr 0 12月 26 20:46 2
-rw-rw-r-- 1 mr mr 0 12月 26 20:46 3
-rw-rw-r-- 1 mr mr 0 12月 26 20:46 4
```

看懂了? 看懂才怪了.

`xargs` 的作用是把输出的内容当做参数传递给下一个命令, 比如:

```
➤ ls | xargs cat
11111
222
33333
444
➤
```

`ls` 的内容是当先文件夹下的文件, 然后 `xargs` 把 `1 2 3 4` 这些 `ls` 输出的东西都传递给了 `cat` ,然后就是上面的效果啦.

我再把作者给的几个例子放到下面, 看聪明的你能不能知道他们是干什么用的呢? 记得不懂的地方问 `man` 哦.

1. `find ~ -name ‘*.log’ -print0 | xargs -0 rm -f`
2. `find /etc -name "*.conf" | xargs ls –l`
3. `cat url-list.txt | xargs wget –c`
4. `find / -name *.jpg -type f -print | xargs tar -cvzf images.tar.gz`
5. `ls *.jpg | xargs -n1 -i cp {} /external-hard-drive/directory`

Sort命令

一般来说命令的命名都是跟命令的作用有关的, 嗯, 我莫名想到了 thefuck ... (逃

`sort` 就是用来排序的, 没错, 给文件内容排序:

```
➤ for i in {1..10};do echo $((RANDOM/20)) >> random; done
➤ cat random
1164
1522
69
1470
1232
642
1494
12
543
685
➤
```

然后我们 `sort` 一下:

```
➤ sort random
1164
12
1232
1470
1494
1522
543
642
685
69
➤ sort -n random
12
69
543
642
685
1164
1232
1470
1494
1522
➤
```

是不是很好玩? 最后一个数字竟然是 69 ... 我好污...

再来个例子:

```
➤ cat passwd
root:x:0:0
daemon:x:1:1
bin:x:2:2
sys:x:3:3
sync:x:4:65534
games:x:5:60
man:x:6:12
lp:x:7:7
mail:x:8:8
news:x:9:9
➤
```

我们先不带任何参数的排一下序:

```
➤ sort passwd
bin:x:2:2
daemon:x:1:1
games:x:5:60
lp:x:7:7
mail:x:8:8
man:x:6:12
news:x:9:9
root:x:0:0
sync:x:4:65534
sys:x:3:3
➤
```

看到了吗, `sort` 按照首字母排序了.

这次我们按照最后一个字段进行排序:

```
➤ sort -t : -k 4 passwd
root:x:0:0
daemon:x:1:1
man:x:6:12
bin:x:2:2
sys:x:3:3
games:x:5:60
sync:x:4:65534
lp:x:7:7
mail:x:8:8
news:x:9:9
```

好像不太对，再加个参数呢：

```
➤ sort -n -t : -k 4 passwd
root:x:0:0
daemon:x:1:1
bin:x:2:2
sys:x:3:3
lp:x:7:7
mail:x:8:8
news:x:9:9
man:x:6:12
games:x:5:60
sync:x:4:65534
```

是不是看起来爽多了？

作者的这个例子很不错：

```
sort -t . -k 1,1n -k 2,2n -k 3,3n -k 4,4n /etc/hosts
127.0.0.1 localhost.localdomain localhost
192.168.100.101 dev-db.thegeekstuff.com dev-db
192.168.100.102 prod-db.thegeekstuff.com prod-db
192.168.101.20 dev-web.thegeekstuff.com dev-web
192.168.101.21 prod-web.thegeekstuff.com prod-web
```

把IP地址排序，可以研究下他是怎么做到的.

最后原作者还介绍了几套组合拳:

- ps –ef | sort #给进程排序
- ls -al | sort -nk 5 #根据文件大小进行排序
- ls -al | sort -nrk 5 #根据文件大小反向排序

注意,这里跟原著有差别哦, 因为我按照原著上的并没有成功, 我的系统是Ubuntu 15.10

Uniq 命令

这个命令也很好理解, 就是去重.

比如我又这样一个文件.

```
➤ cat uniq_file
1
1
2
2
3
3
4
4
5
```

然后用 `uniq` 把里面的内容去重:

```
➤ uniq uniq_file
1
2
3
4
5
➤
```

查看 `man uniq` 获取更多.

Cut 命令

`cut` 命令是用来切割数据的, 没错, 竖着割.

刚才搞到三蛋的库子, 那么我就拿这个数据库做例子吧 :)

首先看文件里有什么:

```
➤ cat pass
ahmed:sales@samaaegy.net:197.44.60.223:Ahmed2542604
ahmed:sales@spctec.com:41.196.193.71:AAHMEDEMAD944405
Ahmed:sales@time2timegroup.com:2.49.209.135:a13c14zI1Nwdck3a
ahmed:sales1@safetymisr.com.eg:197.44.60.223:ahmed2542604
ahmed:salesway@msn.com:216.241.47.198:hggih;fv
Ahmed:salfiste.hacker@gmail.com:41.225.249.155:suffeitula.10@
ahmed:salhexp@gmail.com:217.55.82.202:magicmasterz01
ahmed:salihomer681@yahoo.com:2.89.211.62:123qwe
Ahmed:salikok@live.com:87.109.81.252:faraz54321
ahmed:salma.ali1993@yahoo.com:217.26.255.72:asdf1234
ahmed:salma_love8226@yahoo.com:41.238.81.106:medo0125593126
ahmed:salmad123@gmail.com:81.192.238.123:sonunigam1
ahmed:salmamalikmother@gmail.com:65.49.14.11:love-you2
ahmed:salman.a7med@gmail.com:41.137.61.65:azerty123654789
ahmed:salmankhan200007@gmail.com:65.49.14.71:123asdzxc
ahmed:salmanlakho813@gmail.com:182.182.14.9:2k11swe75
```

好, 里面是以 `用户名:邮箱:IP地址:密码` 这种格式存储的, 那么,如果我们要想提取处里面所有的密码该怎么办呢?

这就是 `cut` 的威力了:

```
➤ cut -d : -f 4 pass
Ahmed2542604
AAHMEDEMAD944405
a13c14zI1Nwdck3a
ahmed2542604
hggih;fv
suffeitula.10@
magicmasterz01
123qwe
faraz54321
asdf1234
medo0125593126
sonunigam1
love-you2
azerty123654789
123asdzxc
2k11swe75
➤
```

详细解释一下这条命令, 其中 `-d` 参数后面跟的是分隔符, 我们看到, 里面的数据都是用 `:` 分隔的, 所以用 `-d :` 把每一列分开, 然后的 `-f` 代表第几列, `f` 也就是 `filed` 的意思. 由于密码是在第4个字段, 所以这里是 `-f 4` .

提取用户名+密码呢?

```
➤ cut -d : -f 1,4 pass #用','分隔我们要选取的字段
ahmed:Ahmed2542604
ahmed:AAHMEDEMAD944405
Ahmed:a13c14zI1Nwdck3a
ahmed:ahmed2542604
ahmed:hggih;fv
Ahmed:suffeitula.10@
ahmed:magicmasterz01
ahmed:123qwe
Ahmed:faraz54321
ahmed:asdf1234
ahmed:medo0125593126
ahmed:sonunigam1
ahmed:love-you2
ahmed:azerty123654789
ahmed:123asdzxc
ahmed:2k11swe75
```

提取邮箱后面的呢?

```
➤ cut -d : -f 2- pass #用'-'表示从哪儿到哪儿,如果不填则表示最后(或者最前)
sales@samaaegy.net:197.44.60.223:Ahmed2542604
sales@spctec.com:41.196.193.71:AAHMEDEMAD944405
sales@time2timegroup.com:2.49.209.135:a13c14zI1Nwdck3a
sales1@safetymisr.com.eg:197.44.60.223:ahmed2542604
salesway@msn.com:216.241.47.198:hggih;fv
salfiste.hacker@gmail.com:41.225.249.155:suffeitula.10@
salhexp@gmail.com:217.55.82.202:magicmasterz01
salihomer681@yahoo.com:2.89.211.62:123qwe
salikok@live.com:87.109.81.252:faraz54321
salma.ali1993@yahoo.com:217.26.255.72:asdf1234
salma_love8226@yahoo.com:41.238.81.106:medo0125593126
salmad123@gmail.com:81.192.238.123:sonunigam1
salmamalikmother@gmail.com:65.49.14.11:love-you2
salman.a7med@gmail.com:41.137.61.65:azerty123654789
salmankhan200007@gmail.com:65.49.14.71:123asdzxc
salmanlakho813@gmail.com:182.182.14.9:2k11swe75
```

`cut` 不仅可以用分隔符来把数据分开, 还可以按照字符分开,比如,我们要提取里面第8个字符,就可以这样:

```
➤ cut -c 8 pass
a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
a
➤
```

或者第2到第8个字符:

```
➤ cut -c 2-8 pass
hmed:sa
hmed:sa
hmed:sa
hmed:sa
hmed:sa
hmed:sa
hmed:sa
hmed:sa
hmed:sa
hmed:sa
hmed:sa
hmed:sa
hmed:sa
hmed:sa
hmed:sa
hmed:sa
➤
```

虽然上面的数据看起来没有意义，但 `cut` 用好了会节省你很多的时间.

Stat 命令

`stat` 命令是用来查看文件详细信息的:

```
➤ stat pass
  File: ‘pass’
  Size: 870           Blocks: 8          IO Block: 4096   regula
r file
Device: fc01h/64513d    Inode: 5013885     Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/      mr)   Gid: ( 1000/
     mr)
Access: 2015-12-27 20:31:13.127178405 +0800
Modify: 2015-12-27 20:31:10.959155945 +0800
Change: 2015-12-27 20:31:10.959155945 +0800
 Birth: -
➤
```

扩展阅读:

Stat - 获取文件信息

Diff 命令

`diff` 是用来找不同的 :)

使用很简单:

`diff [options] file1 file2`

比如我又两个文件:

```
➤ cat diff1
1
2
a
➤ cat diff2
1
2
3
4
➤
```

然后我 `diff` 一下:

```
➤ diff diff1 diff2
3c3,4
< a # < 表示在第二个文件中缺失的部分
---
> 3 # > 表示在第一个文件中缺失的部分
> 4
➤
```

上面的结果中:

`---` 上面的内容是 `diff1` 文件的, `---` 下面的内容则是 `diff2` 文件的内容.

那么 `3c3,4` 代表的是什么?

stackexchange上面有人说可以忽略...

但是第三个答案说的就比较详细:

> `3d2` and `5a5` denote line numbers affected and which actions were performed. `d` stands for deletion, `a` stands for adding (and `c` stands for changing). the number on the left of the character is the line number in file1.txt, the number on the right is the line number in file2.txt. So `3d2` tells you that the 3rd line in file1.txt was deleted and has the line number 2 in file2.txt (or better to say that after deletion the line counter went back to line number 2). `5a5` tells you that the we started from line number 5 in file1.txt (which was actually empty after we deleted a line in previous action), added the line and this added line is the number 5 in file2.txt.

但其实知道了也没卵用... 不如 `vimdiff` 或者 `diff -u` 更直观.

扩展阅读:

- About diff
- Vimdiff

Ac 命令

先要安装`acct` - `#sudo apt-get install acct`

然后...

1. `ac -d` 显示当先登录用户的使用时间
2. `ac -p` 显示所有用户的登陆使用时间
3. `ac -d xxxx` 显示xxx用户的登陆使用时间

感觉用处不是很大, 除非你的机子是多用户共同操作的.

有一种情况除外, 可以查看有没有未经允许的登陆, 被黑了好查证, 不过... 被黑之后这种文件一般都不存在了...

让命令在后台执行

后台执行命令有很多方法,比如:

1. 使用 `&`

在你想要后台执行的命令最后加上一个 `&` 就能让命令在后台运行了.

```
➤ htop &
[1] 12781
➤ ps
  PID TTY          TIME CMD
12554 pts/5    00:00:00 bash
12781 pts/5    00:00:00 htop
12782 pts/5    00:00:00 ps
➤
```

怎样把再它调出来呢? 用 `fg` 命令. 还可以用 `jobs` 命令查看当前终端上运行的后台程序.

2.使用 `nohup`

> nohup - run a command immune to hangups, with output to a non-tty

这是正统的后台运行.

`nohup ./my-shell-script.sh &`

默认程序的输出内容会保存到当前目录下的 `nohup.out` 文件中, 所以一般运行的时候都是这样(如果你不需要查看运行结果的话):

`nohup ./my-shell-script.sh &> /dev/null &`

3.使用 `screen`

这也是后台运行常用的工具，比起 `nohup`，它可以随时随地查看运行情况，并且进行操作，更多关于 `screen` 的介绍请看：Linux 技巧：使用 screen 管理你的远程会话

其实还有一个比 `screen` 更酷炫的软件，叫 `tmux`，有兴趣可以去了解下哦~

4. 使用 `at`

这真是巧妙，作者能想到这一点我也是佩服，`at` 一般都是作为计划任务来使用的，但是这样也能起到后台运行的效果.

再明天上午十点执行备份脚本：`at -f backup.sh 10 am tomorrow`

或者现在就执行：`at -f backup.sh now` **# -f** 的意思是执行后面的文件

查看任务队列用 `atq`

删除任务可以用 `adrm [number_of_task]` 或者 `ad -d [number_of_task]`

5.使用 `watch`

这个说起来就有点牵强了，毕竟 `watch` 不是干这个的呀.

原作说的是持续运行某条命令，凑活写上吧..

每隔五秒执行一次 `df -h`，如果不加 `-n 5` 的话，默认是两秒，`watch -n 5 'df -h'`

扩展阅读

- bg, fg, &, Ctrl-Z – 5 Examples to Manage Unix Background Jobs
- Unix Nohup: Run a Command or Shell-Script Even after You Logout
- Screen Command Examples: Get Control of Linux / Unix Terminal
- at, atq, atrm, batch Commands using 9 Examples

Sed 替换基础

这里介绍了 `sed` 的替换技巧

首先看一下用法:

```
sed 's/REGEXP/REPLACEMENT/FLAGS' filename
```

其中:

- **s** 表示执行替换操作
    - 常用的还有 **d** 表示删除
- **/** 表示分隔符
    - 分隔符还可以用 **@** 或者 **%** 或者 **;** 或者 **:** 表示
- **REGEXP** 表示匹配的正则
- **REPLACEMENT** 表示要替换的内容
- **FLAGS** 表示标志
    - **g** 全局替换, 从头到尾都换
    - **n** 可以是任意数字, 表示的是替换第n次出现的地方
    - **p** 打印两次匹配到的内容, 即,如果替换成功则打印两遍
    - **i** 忽略大小写的匹配
    - **w** 后面跟个文件, 如果产生了替换,则将替换后的结果输出到文件中

然后我们来看这样一个文件:

```
➤ cat sed.txt
# Instruction Guides
1. Linux Sysadmin, Linux Scripting etc.
2. Databases - Oracle, mySQL etc.
3. Security (Firewall, Network, Online Security etc)
4. Storage in Linux
5. Productivity (Too many technologies to explore, not
much time available)
#
Additional FAQS
6. Windows- Sysadmin, reboot etc.
```

1. 把第一个 'Linux' 替换成 'Linux-Unix'

```
➤ sed 's/Linux/Linux-Unix/' sed.txt
# Instruction Guides
1. Linux-Unix Sysadmin, Linux Scripting etc.
2. Databases - Oracle, mySQL etc.
3. Security (Firewall, Network, Online Security etc)
4. Storage in Linux-Unix
5. Productivity (Too many technologies to explore, not
much time available)
#
Additional FAQS
6. Windows- Sysadmin, reboot etc.
```

这个例子很简单, 不多说了.

2. 把 'Linux' 全部替换成 'Linux-Unix'

```
➤ sed 's/Linux/Linux-Unix/g' sed.txt #注意,这里是全部替换哦,所以有个g
(global)
# Instruction Guides
1. Linux-Unix Sysadmin, Linux-Unix Scripting etc.
2. Databases - Oracle, mySQL etc.
3. Security (Firewall, Network, Online Security etc)
4. Storage in Linux-Unix
5. Productivity (Too many technologies to explore, not
much time available)
#
Additional FAQS
6. Windows- Sysadmin, reboot etc.
```

3.仅把第二个 'Linux' 换成 'Linux-Unix'

```
➤ sed 's/Linux/Linux-Unix/2' sed.txt
# Instruction Guides
1. Linux Sysadmin, Linux-Unix Scripting etc.
2. Databases - Oracle, mySQL etc.
3. Security (Firewall, Network, Online Security etc)
4. Storage in Linux
5. Productivity (Too many technologies to explore, not
much time available)
#
Additional FAQS
6. Windows- Sysadmin, reboot etc.
```

看到了吗，这里的标志位是 `2` 哦~

4. 全部替换+打印替换内容到屏幕+保存输出到文件

```
➤ sed -n 's/Linux/Linux-Unix/gpw output' sed.txt
1. Linux-Unix Sysadmin, Linux-Unix Scripting etc.
4. Storage in Linux-Unix
------
➤ cat output
1. Linux-Unix Sysadmin, Linux-Unix Scripting etc.
4. Storage in Linux-Unix
```

这次的例子有点奇怪，标志位是 `gpw` 我们已经知道 `g` 代表的是全部替换,那么 `p` 和 `w` 又是啥？通过 `info sed` 我们可以查到：

> `p' - If the substitution was made, then print the new pattern space.
>
> `w FILE-NAME' - If the substitution was made, then write out the result to the named file.

现在你懂了嘛？不懂？查词典去!

5.仅仅替换特殊的行

```
➤ sed '/\-/s/\-.*//g' sed.txt
# Instruction Guides
1. Linux Sysadmin, Linux Scripting etc.
2. Databases
3. Security (Firewall, Network, Online Security etc)
4. Storage in Linux
5. Productivity (Too many technologies to explore, not
much time available)
#
Additional FAQS
6. Windows
```

看到这个命令与其他的不同了吗?

没错, 就是在 `s` 前面加了点东西, 加的东西就是我们要匹配的行, 这次加的是 `\-` , 所以就替换了 `-` 后面的内容. 举一反三, 你是不是可以想出别的方案呢? 试试看!

6.删除某些字符

```
➤ sed 's/\(.....\)\(...\)$/\2/g' sed.txt
# Instructiodes
1. Linux Sysadmin, Linux Scripttc.
2. Databases - Oracle, mytc.
3. Security (Firewall, Network, Online Securtc)
4. Storage nux
5. Productivity (Too many technologies to explnot
much time avle)
#
AdditioAQS
6. Windows- Sysadmin, rebtc.
```

看懂了吗? 没有? 好, 那看下一个:

```
➤ sed 's/...$//g' sed.txt
# Instruction Gui
1. Linux Sysadmin, Linux Scripting e
2. Databases - Oracle, mySQL e
3. Security (Firewall, Network, Online Security e
4. Storage in Li
5. Productivity (Too many technologies to explore,
much time availab
#
Additional F
6. Windows- Sysadmin, reboot e
```

这个应该看懂了吧?

上一个例子是删除了最后三个字符,上上一个例子是删除了倒数第八到第五个字符,也就是,把最后八个字符换成了最后三个字符,有点绕? 多想一会儿!

再来一个删除html标签的:

```
➤ echo "This <b> is </b> an <i>example</i>." | sed -e 's/<[^>]*>
//g'
This  is  an example.
```

最后一个例子:

```
➤ sed 's/#.*//g' sed.txt

1. Linux Sysadmin, Linux Scripting etc.
2. Databases - Oracle, mySQL etc.
3. Security (Firewall, Network, Online Security etc)
4. Storage in Linux
5. Productivity (Too many technologies to explore, not
much time available)

Additional FAQS
6. Windows- Sysadmin, reboot etc.
```

替换显得太难看了,不如直接删除:

```
➤ sed '/#.*/d' sed.txt
1. Linux Sysadmin, Linux Scripting etc.
2. Databases - Oracle, mySQL etc.
3. Security (Firewall, Network, Online Security etc)
4. Storage in Linux
5. Productivity (Too many technologies to explore, not
much time available)
Additional FAQS
6. Windows- Sysadmin, reboot etc.
```

`sed` 是个好东西, 光 `info sed` 就有98K这么大, 好好看看咯~ (我没看完...逃...

扩展阅读

- Advanced Sed Substitution Examples
- Sed Tutorial

Awk 简介

简介

> **Awk** stands for the names of its authors “Aho, Weinberger, and Kernighan”

Awk也是一种编程语言, 他能让你格式化数据以及生成特定格式的报告.(翻译的好糙...)

总之, 你只需要记住它是用来格式化数据的就OK了.

`Awk` 更像是一个过滤器, 它把读入的数据一行一行的过滤, 匹配你想要的内容, 如果匹配到了, 那就格式化输出, 匹配不到的就忽略.

`Awk` 有一些特性:

- 它把数据视为 '记录' 和 '字段'
- 它也有变量, 条件和循环.
- 它有数学操作符和字符串操作符
- 它能生成格式化的报告
- 它从标准输入读取数据, 把过滤之后的数据从标准输出打印出来, 不处理空文件.

基本语法:

```
awk '/search pattern1/ {Actions} /search pattern2/ {Actions}' fi
le
```

其中:

- `search pattern` 是要匹配的字符串
- `Actions` 是匹配到之后所进行的动作
- 它能处理多个匹配和行为
- `file` 是值输入文件
- 单引号的作用是防止里面的特殊字符被shell处理

Awk的工作方法

1. `Awk` 每次从输入文件中读取一行.
2. 对于每一行, 如果匹配到了相应的内容, 就会执行相应的动作.
3. 如果匹配不到, 就什么也不做.
4. 在上面的语法中, 匹配项和动作有一即可.
5. 如果没有给出匹配项, Awk就会把每一行按照给定的动作执行
6. 如果动作没有给出, 默认是打印.
7. 如果大括号里面是空的, 那就什么也不干.(因为这样就代表给出了一个空的动作)
8. 在大括号里的每一个动作需要用分号(`;`)隔开.

然后我们开始举栗子:

先是有这样一个文件:

```
➤ cat awk.txt
100 Thomas Manager Sales $5,000
200 Jason Developer Technology $5,500
300 Sanjay Sysadmin Technology $7,000
400 Nisha Manager Marketing $9,500
500 Randy DBA Technology $6,000
```

打印每一行

```
➤ awk '{print;}' awk.txt
100 Thomas Manager Sales $5,000
200 Jason Developer Technology $5,500
300 Sanjay Sysadmin Technology $7,000
400 Nisha Manager Marketing $9,500
500 Randy DBA Technology $6,000
➤
```

这里我们没有给出匹配项, 仅给出了动作, `print` 打印.

打印匹配到的行

```
➤ awk '/Thomas/;/Nisha/' awk.txt
100 Thomas Manager Sales $5,000
400 Nisha Manager Marketing $9,500
```

这里跟原著有点不太一样，原著是用回车把每一个要匹配的内容分开的，但是这样不好修改，看起来也怪怪的，所以，不如直接用分号把他们隔开啦~

上面的栗子打印了匹配到那两个名字的行(记录).

打印特定的字段(列)

这个例子打印了第2和第5个字段.

```
➤ awk '{print $2,$5}' awk.txt
Thomas $5,000
Jason $5,500
Sanjay $7,000
Nisha $9,500
Randy $6,000
```

再看下面这个例子:

```
➤ awk '{print $3,$NF}' awk.txt
Manager $5,000
Developer $5,500
Sysadmin $7,000
Manager $9,500
DBA $6,000
```

有点不一样了是吧? `$NF` 的意思是最后一个字段.

每个字段都要用逗号(`,`)分开.

开始和结束

`awk` 有一种特定的语法，开始和结束.

```
BEGIN { Actions}
{ACTION} # 文件中的每一行默认的动作
END { Actions }
```

有什么用呢? 正如表述所言, `开始` 定义了输出一开始的动作,而 `结束` 则定义了输出结束时的动作.

比如:

```
➤ awk 'BEGIN {print "Name\t Designation\tDepartment\tSalary";} {
print $2,"\t",$3,"\t",$4,"\t",$NF;} END{print "Report Generated\
n--------------";}' awk.txt
Name      Designation     Department     Salary
Thomas       Manager       Sales       $5,000
Jason       Developer       Technology       $5,500
Sanjay       Sysadmin       Technology       $7,000
Nisha       Manager       Marketing       $9,500
Randy       DBA       Technology       $6,000
Report Generated
--------------
```

如果太长了, 中间可以用回车隔开:

```
$ awk 'BEGIN {print
"Name\tDesignation\tDepartment\tSalary";}
> {print $2,"\t",$3,"\t",$4,"\t",$NF;}
> END{print "Report Generated\n--------------";
> }' employee.txt
Name Designation Department Salary
Thomas Manager Sales $5,000
Jason Developer Technology $5,500
Sanjay Sysadmin Technology $7,000
Nisha Manager Marketing $9,500
Randy DBA Technology $6,000
Report Generated
--------------
```

条件语句

`awk` 也有条件语句, 比如比较大小:

```
➤ awk '$1 >= 200' awk.txt
200 Jason Developer Technology $5,500
300 Sanjay Sysadmin Technology $7,000
400 Nisha Manager Marketing $9,500
500 Randy DBA Technology $6,000
```

上面的例子是打印第一个字段大于等于200的行.

再看下面这个例子, 是对字符串进行比较的:

```
➤ awk '$4 ~ /Tech/' awk.txt
200 Jason Developer Technology $5,500
300 Sanjay Sysadmin Technology $7,000
500 Randy DBA Technology $6,000
```

有点像 `[[ ]]` 里面的正则匹配, 好像就是正则匹配...

```
➤ awk '$4 ~ /[tT][abcde]/' awk.txt
200 Jason Developer Technology $5,500
300 Sanjay Sysadmin Technology $7,000
500 Randy DBA Technology $6,000
```

有点意思了,是吧!

计数!

`awk` 既然是一种编程语言, 那基本的计数也是应该的:

```
➤ awk 'BEGIN { count=0;} $4 ~ /Tech/ { count++; } END { print "Number of employees in Technology Dept=",count;}' awk.txt
Number of employees in Technology Dept= 3
```

```
➤ awk 'BEGIN { count=0;}
> $4 ~ /Tech/ { count++; }
> END { print "Number of employees in Technology Dept=",count;}'
 awk.txt
Number of employees in Technology Dept= 3
```

附言

`awk` 作为一门编程语言, 这点介绍仅仅是皮毛而已, 就像, 虽然你学了点C语言,能输出个12345了, 可C能做的可不是这么简单. :)

(我也只是学了点皮毛...)

扩展阅读

- Awk 入门
- 理解Awk变量
- 8个Awk内置变量
- 7个Awk操作符
- Awk数组

(这些扩展阅读也都是原作者自己写的, 很厉害的一个家伙!

VIM 基本入门(移动)

为什么叫 基本入门 而不是 入门 呢?

因为我觉着我都还没入门...

这里作者主要说的是怎样移动你的光标,毕竟没有鼠标,所以快捷键就显得非常重要了,这里介绍了很多快速移动的快捷键,希望大家熟记于心.(拿出来装逼也好呀)

作者写了八个方面,在此列举:

1. 行间移动
2. 屏幕间移动
3. 单词移动
4. 特殊移动
5. 段落间移动
6. 搜索移动
7. 代码间移动
8. 从命令行中移动(command 模式)

由于这些操作都是需要自己动手的,没办法展示出来(其实还是由办法的,比如gif图,有好多软件可以做到,看这里的回答;还有就是用 `script` "录"下来,不过那个文件还要传到上面,反正挺麻烦的,不如让各位看官自己动手了,好了,不废话了...)

1. 行间移动

按键	方向
k	向上移动
j	向下移动
h	向左移动
l	向右移动
10j	下移**10**行
5h	左移**5**个字母
0	移动到行首
^	移动到行首第一个单词
$	移动到行尾
g_	移动到行尾第一个单词

2. 屏幕间移动

也就是以屏幕为单位的移动啦

按键	方向
H	移动到本屏首行
M	移动到本屏的中间
L	移动到本屏的尾行
Ctrl + f	向上移动一个屏幕
Ctrl + b	向下移动一个屏幕
Ctrl + d	向下移动半个屏幕
Ctrl + u	向上移动半个屏幕

3. 特殊移动

下面的是比较特殊的移动方式:

按键	方向
N%	移动到文件的N%的位置，比如 `50%`
NG	移动到文件第N行，比如 `6G`
gg	移动到文件头
G	移动到文件末尾
`"	移动到上次在"Normal"模式下关闭文件时的地方
`^	移动到上次在"Insert"模式下关闭文件时的地方

4. 单词间移动

按键	方向
e	移动到单词末尾
E	移动到大单词末尾
b	移动到上一个小单词
B	移动到上一个大单词
w	移动到下一个小单词
W	移动到下一个大单词

大小写的区别:

- 大写的移动：移动的单词为一连串. 比如 `192.168.1.1` – 是**1**个大写的单词
- 小写的移动：移动的单词以非数字或字母为分界线. 比如 `192.168.1.1` – 是**7**个小写的单词

其实你自己动手操作下就能够知道他们之间的区别了.

5. 段落间移动

所谓"段落",就是用空行隔开的句子段.

按键	方向
{	移动到段首
}	移动到段尾

6. 搜索移动

按键	方向
/text	从光标处向下搜索
?text	从光标处向上搜索
*****	移动到光标所在单词的下一个位置
#	移动到光标所在单词的上一个位置

其实这里说的不全, 因为还有 `n` 和 `N` 的存在. `n` 是移向下一个搜索目标, `N` 是移向上一个搜索目标.

`*` 就好像一个组合键, 搜索光标所在位置的单词的同时,又移向了下一个目标. 这对于编辑html文档很有用, 闭合标签嘛

`#` 类似, 只不过搜索方向不同而已.

7. 代码间移动

这个就是在两个括号之间移动.... 按 `%` 就可以在两个半闭合的括号来回移动, 所谓"代码间移动".... 我都觉着这名字很狗血...

8. 从命令行中移动(command 模式)

这个的意思是在打开的时候就移动到某一行, 我不知道怎样起名字,暂且称之为标题中的吧... 狗血就狗血吧...

- `vim +10 /etc/passwd` # 打开 `/etc/passwd` 之后,光标在第十行
- `vim +/install README` # 打开README之后,光标在第一个 `install` 前面(如果有的话)
- `vim +?bug README` # 打开README之后, 光标在最后一个 `bug` 前面(同上)

扩展阅读

- 8 Essential Vim Editor Navigation Fundamentals
- Vim search and replace – 12 powerful find and replace examples.
- How To add bookmarks inside the Vim editor
- How To record and play inside the Vim editor

- Correct spelling mistakes automatically inside the Vim Editor

Chmod 命令

感觉作者是在凑数.... `chmod` 还不如 `chattr` 有趣...

三个代表

文件的权限有三个代表, 代表最广大人民....

上面当然是在扯淡, 因为这里没啥好讲的, 每个文件都有自己的属性, 每个属性都有不同的权限, 权限分为三种, 每种代表不同的用户.

- u 代表用户,也就是文件的所有者 (user)
- g 代表用户组, 也就是文件的所有组 (group)
- o 代表其他人, 也就是除上面两者之外的人或者用户组 (others)

三种权限

不同的人,组, 对文件拥有不同的读写权限.

- r 代表读权限 (read) = 4
- w 代表写权限 (write) = 2
- x 代表执行权限 (excute) = 1

如果一个文件是可执行文件, 那么给相应的用户添加 `x` 后,那个用户就可以执行这个文件. 我们知道,目录也是文件, 而目录的可执行权限就是进入目录(或者读取目录的内容,或者对目录里的内容进行操作,比如,删除.), 如果取消了你对某个目录的可执行权限以后, 你就进不去了, 另外一个误区就是, 如果你对某个目录具有执行权限, 那么你就可以对目录下的内容进行移动, 删除操作, 不管这个文件是属于谁的.

上面的 `4 2 1` 分别对应某种权限的数字表示, 我们常说的设置权限为 `755` ,就是让 `u` 的权限为 `7` , `g` 的权限为 `5` , `o` 的权限也为 `5` , `7` 代表什么呢? 代表 `4+2+1` 也就是 `rwx` 权限, 那么 `5` 也就好解释了, `5=4+1` ,也就是 `r+x` 权限.

下面开始翻译...

1. 给文件的所有者添加执行权限: `chmod u+x filename`
2. 给文件所有者添加读权限,并且给文件所属组添加执行权限: `chmod u+r,g+x filename`

3. 给文件的所有者去除读权限和执行权限: `chmod u-rx filename`
4. 给所有用户(u+g+o)添加文件的执行权限: `chmod a+x filename`
5. 设置某个文件(file2)的权限与另一个文件(file1)相同: `chmod --reference=file1 file2`
6. 递归设置文件权限: `chmod -R 755 dir/`
7. 匹配正则: `chmod u+x *.py`

扩展阅读

Beginners Guide to File and Directory Permissions

我感觉这里啰嗦的东西完全是在凑命令... 建议大家看一下比较冷门的 `chattr` , 算式隐藏命令吧, 尤其是 `a` 和 `i` 这两个权限. 很有用的.

同时用 `tail` 查看两个log文件

原作者在这里啰里啰嗦的说了一堆, 总结起来就一个技巧:

```
tail -f access.log -f error.log
```

这就是用 `tail` 同时查看两个增长的文件, 这样你就不用开启两个终端,或者是用 `screen` or `tmux` 这些额外工具了.

用处还是蛮大的. 比如在本地调试网站的时候, 这个命令就很有用, 同时查看好几个日志.

Less 命令

查看文件的时候，如果文件不是很小(超过了ternimal的高度)，那最好还是用 `more` 或者 `less` 来查看.

`less` 和 `more` 的区别在于，`more` 只能往下翻，而 `less` 允许用户往上翻 :)

而且用 `less` 查看文件的时候，`less` 并不是把文件全部加载到内存中后再输出,而是直接就输出，相当给力. 如果你又一个大文件(超过1G)，你想简单地浏览一下时，还是推荐用 `less` .无论从资源消耗还是打开速度，`less` 绝对是你的不二选择.

less 搜索移动

我们可以在用 `less` 打开的文件中搜索内容，命令语法跟 `vim` 差不多，都是用 `/` 向下搜索，`?` 向上搜索. `n` 搜索下一个，`N` 搜索上一个.

这里有个小技巧，推荐用 `?` 来搜索，因为这样可以不用转义 `/`，如果你搜索的内容正好有这个字符的话.

less 翻页

我们当然可以使用 `PageUp` 或者 `PageDown` 来翻页，不过假如你足够懒，不想让手指移动很多路径，那么你就可以选择下面的方案:

- `Ctrl + f` 或者 `f` 向前翻一页
- `Ctrl + b` 或者 `b` 向后翻一页
- `Ctrl + d` 或者 `d` 向下翻半页
- `Ctrl + u` 或者 `u` 向上翻半页

less 移动

跟 `vim` 一样，`hjkl` 方向移动，当然,方向键和鼠标滚轮也都好使.

`G` -移动到末尾，`g` -移动到文件头部，`q` 或 `ZZ` 退出.

`10j` 往下移动10行，`5k` 往上移动5行.

模仿 `tail -f`

没错, less强大到它可以追踪文件流, 按下 `F` 后, 就可以像 `tail -f` 一样查看文件的变化, `Ctrl + c` 可以退出.

其他

`Ctrl + g` 显示当前进度, 文件信息(行数,字节数)

`v` - 这个特别有用, 如果你查看过这个文件后想用你默认的编辑器编辑一下这个文件的话, 那么按一下 `v` 就可以了~ 很方便.

`h` 显示帮助, 包括各种快捷键的详细介绍.

`&pattern` 显示匹配到pattern的行, 正则表达式哟.

标记

如果你浏览到某个地方想要标记一下, 那么按下 `m` 键后再按下一个标记键,比如, `a` , 那么你就在当前屏幕有了一个名字叫 `a` 的标记点(区分大小写的哦, **`a`** 和 **`A`** 是不一样的标记点!).

那怎样返回这个标记呢? 再按下 `'` ,也就是单引号, 底部就会出现 `goto mark:` 这样的提示, 按下 `a` 就回到了 `a` 标记点, 按下 `A` 就会到 `A` 标记点...

然后, 这个跟 `VIM` 一模一样!!! (`vim` 里面标记也是这样的, 还能输入 `marks` 来查看所有的标记.)

多文件操作

你可以用 `less Textfile Logfile` 来同时打开两个文件,文件之间的切换用 `:n` 和 `:p` , `n` 代表 `next` ,下一个文件; `p` 呢, 代表 `previous` , 上一个文件.

当然, 在浏览文件的同时也可以打开另一个文件, 输入 `:e` ,然后就会提示: `Examine:` 让你输入文件名, 文件名是可以用 `Tab` 补全的.

扩展阅读

- Less Command: 10 Tips for Effective Navigation
- Open & View 10 Different File Types with Linux Less Command

Wget 下载器

`wget` 是Linux标配的下载器, 虽然作者说他是Linux上最好的选择, 但是还有 `aria2` 这个多线程下载神器, 个人感觉要比 `wget` 好.

在这里, 作者介绍了15个 `wget` 的应用.

1. 下载单一文件

```
wget "http://url.com/file"
```

很简单 :)

2. 将下载的文件重命名

```
wget "http://url.com/file" -O rename
```

`-O` 的选项, 使得下载的文件被重命名了.这个开关很有用, 因为 `wget` 默认下载的文件是根据URL来命名的, 如果URL中有很多参数, 那么你下载的文件就会是个乱七八糟的名字, 最常见的是下载百度云...(百度云不开会员就限速!)

3. 下载限速

我们可以用 `--limit-rate` 这个选项来限制 `wget` 的下载速度, 比如:

```
wget --limit-rate=200k "http://url.com/file.tar.bz2"
```

就会限制下载速度到 200KB/s

4. 继续下载

这个功能也很常用, 比如, 正下载到一半断网了, 或者网络中断, 那么就可以用 `-c` 的开关来定义继续下载, continue嘛!

需要注意的是，如果不加这个开关，那么就会重新下载一个新的文件，并且如果当前目录下有重名的文件，会在重名文件的后缀名上加一个 `.1`，如果有 `.1` 了，就会加个 `.2`，所以这个 `-c` 的开关很重要呢！

```
wget -c "http://url.com/file.tar.bz2"
```

5. 后台下载

这个也很有用，甬道的开关是 `-b` - background

如果开启了这个开关的话，就不会有任何输出，但是记录还是要有的，默认是在当前目录下创建一个 `wget-log` 的文件，若你需要指定记录文件的名字呢，就需要用到 `-o` 这个选项了，后面接要保存的文件名，注意了，这里是小写的 `o` 哦！

```
wget -b "http://url.com/downloads.tar" -o downloads.log -O download_file"
```

6. 指定下载的 User Agent

用 `--user-agent` 这个选项可以手动指定我们的 user-agent，这在目标服务器限制User Agent的时候很有用。

```
wget --user-agent="I'm not Wget.." "http://url.com/downloads.tar"
```

7. 测试目标文件

我们拿到一个下载地址之后就直接开始下载嘛？一般情况下是的，但是不排除我们只想看一下目标文件是什么，类型啦，大小啦，甚至目标文件是否真的存在(返回代码是什么).

这个时候就要用到另一个选项 `--spider`，没错，就是蜘蛛，先让蜘蛛去爬一下，看看我们要下载的文件成色如何。

```
➤ wget --spider "localhost/user.zip" -O /dev/null
Spider mode enabled. Check if remote file exists.
--2016-01-03 20:21:11--  http://localhost/user.zip
Resolving localhost (localhost)... 127.0.0.1
Connecting to localhost (localhost)|127.0.0.1|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3463466 (3.3M) [application/zip]
Remote file exists.

➤
```

上面的就是我们的蜘蛛给我们返回的内容, 比较详细呢!

在看一个错误的(404):

```
➤ wget --spider rabbit/index.html
Spider mode enabled. Check if remote file exists.
--2016-01-03 20:15:48--  http://rabbit/index.html
Resolving rabbit (rabbit)... 127.0.0.1, 127.0.1.1
Connecting to rabbit (rabbit)|127.0.0.1|:80... connected.
HTTP request sent, awaiting response... 404 Not Found
Remote file does not exist -- broken link!!!

➤
```

然后服务器端的日志为:

```
127.0.0.1 - - [03/Jan/2016:20:15:48 +0800] "HEAD /index.html HTTP/1.1" 404 0 "-" "Wget/1.16.1 (linux-gnu)"
```

看到了吗, spider实质上是向服务器发送了一个HEAD请求. 以此查看目标文件的信息.

这样做的用途, 作者也给了示例, 感觉最有用的还是检查书签, 检查一下你收藏栏里的网址是否还健在.

8. 设定重试次数

有时候网络不好, 下载经常重试, 所以就需要设定retry的次数, 可以用 `-tries` 这个选项来定义:

```
wget --tries=75 "http://url.com/file.tar.bz2"
```

9. 批量下载(从文件导入下载链接)

这个功能实际上是从一个文件读取下载链接, 然后慢慢下载...

比如, 有个 url.txt 里面存放着很多下载地址, 那么就可以用 `-i` 这个参数来定义:

```
wget -i url.txt
```

在此安利一波我的杂货仓: http//ipv6.kfd.me 仅限IPv6地址访问.

10. 下载整个网站(爬爬爬)

```
wget --mirror -p --convert-links -P ./LOCAL-DIR WEBSITE-URL
```

这个... 先一一介绍参数吧:

- `--mirror` 打开'镜像储存'开关 因为我们是复制整个网站嘛
- `-p` 下载所有相关文件 为了展示HTML页面所必须的文件,css,js,img啥的
- `–convert-links` 下载完成之后, 把原始链接转换成相对连接(根据本地环境)
- `-P ./LOCAL-DIR` 保存到指定的目录下

这个例子实际上就是爬站... 或者说, 窃.

11. 拒绝下载某种文件

再上面的例子中, 如果你不想下载某种特定类型的文件, 就可以用 `--reject` 参数来指定要拒绝的文件类型, 比如 `--reject=gif` 就拒绝下载gif文件.

12. 保存下载日志

上面(第五点)我已经提过了, 在此不再赘述,

参数为 `-o` 小写.

13. 超过设定大小后自动退出

如果你想让下载的文件不超过20M，就可以指定 `-Q` 的参数，如果超过多少兆之后，就退出下载。

```
wget -Q5m -i url.txt
```

这个参数生效的条件是，url必须都要从文件中读取。

14. 仅下载特定类型的文件

用途：

- 下载某个网站上的所有图片
- 下载某个网站上的所有视频
- 下载某个网站上的所有PDF文件

```
wget -r -A.pdf "http://url-to-webpage-with-pdfs.com/"
```

`-r` 表示递归下载，默认深度为5。

15. FTP登陆下载

```
wget --ftp-user=USERNAME --ftp-password=PASSWORD "DOWNLOAD.com"
```

`wget` 作为一个优秀的下载软件，功能强大到难以想象，想知道更多？问一下 `man` 吧！

第三章 - SSH技巧

整本书进展到 28% 了, 还不算慢, 照这样下去, 再有一周就可以搞定了 :)

~~顺便提一下我毕不了业的问题, 以后看到这里也算有个念想...... 2016-1-3~~

这一章介绍了一些关于SSH命令的小技巧和冷门选项.

虽然不是很有用, 好多我在本地也没成功, 但是知道的多了也不是一件坏事.(就像我挂过的科目...

如果看不下去了就跳过吧, 实际意义不是很大. (除了第一个.)

调试ssh客户端

这里就讲了查看ssh连接的过程，打开 `-v` 的开关就好了.

```
➤ ssh root@localhost -v
OpenSSH_6.9p1 Ubuntu-2, OpenSSL 1.0.2d 9 Jul 2015
debug1: Reading configuration data /home/mr/.ssh/config
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: /etc/ssh/ssh_config line 19: Applying options for *
debug1: auto-mux: Trying existing master
debug1: Control socket "/tmp/root@localhost:22" does not exist
debug1: Connecting to localhost [127.0.0.1] port 22.
debug1: connect to address 127.0.0.1 port 22: Connection refused
ssh: connect to host localhost port 22: Connection refused
```

过程很详细.

SSH逃逸字符

说真的，我没有操作成功，报错信息为：

`escape not available to multiplexed sessions`

作者的意图是，将ssh作为一个任务，运行在后台，随时切换，这当然是好的，可是在我们知道了 `screen` 以及 `tmux` 这些软件之后，就不需要在本地和远程切换了.不过为了尊重原著，还是在此翻译一下：

1. 登陆远程系统：`localhost$ ssh -l jsmith remotehost`
2. 现在我们在远程的系统上了：`remotehost$`
3. 挂起当前任务，按下 `~` 之后再按 `Ctrl+Z` .

```
remotehost$ ~^Z
[1]+ Stopped ssh -l jsmith remotehost
localhost$
```

PS：一开始的时候要按两个 `~~` ，问我原因？看这里：
https://lonesysadmin.net/2011/11/08/ssh-escape-sequences-aka-kill-dead-ssh-sessions/

4.现在就会到本地目录了：

```
localhost$ jobs
[1]+ Stopped ssh -l jsmith remotehost
```

5.当你需要返回的时候,就可以这样：

```
localhost$ fg %1
ssh -l jsmith remotehost
remotehost$
```

显示SSH会话状态

跟上一个一样, 仍旧没成功, 估计你也一样... 可能是因为版本不同吧.

`~s` 显示会话状态.

跳过.

OpenSSH安全配置

OpenSSH的配置文件为 `/etc/ssh/sshd_config`

下面介绍了7个需要更改的地方, 也不能说是需要, 只能说, 这样做会让你的主机在互联网中更安全.

在上面那个配置文件中, 有好多行的开始是一个 `#` 字符, 这表示这个行是一个注释, 而里面的配置则是默认的, 也让就是说, 注释掉的行都是默认配置, 你可以把注释取消, 然后更改选项.

禁止root账户登录

服务器默认是允许root登陆的, 但是最好不要允许root直接登陆, 而是用你的账户登陆, 然后 `su -` 来进入root账户.

(其实这一点我本人不敢苟同, 也有可能是因为手里的服务器都是自己掌控, 所以不分你我, 但我想大多数人也就一台自用的服务器, 这个时候就无所谓那个账户了, 而且在后面开启证书验证, 关闭密码登陆的时候, 服务器是相当安全的(就非法登陆而言,漏洞除外), 所以, 如果你只有一台机子而这台机子就你自己用的话, 还是忽略这点吧.)

之所以这么做, 是因为, 如果允许root登陆, 那么每一个以root登录的用户在进行一系列操作之后, 无法查出具体是谁做的, 也就是, 翻了错误可以丢锅... 但是当禁止root登陆之后, 每个用户要想行使root权限, 就必须使用 `su -` , 这一切都是记录下来的 :) 所以犯了错就有据可查咯~

具体操作:

```
$ vi /etc/ssh/sshd_config
PermitRootLogin no #修改这个地方
```

仅允许特定用户/组登陆

具体操作:

```
$ vi /etc/ssh/sshd_config
AllowUsers ramesh john jason #允许登陆的用户
AllowGroups sysadmin dba #允许登陆的用户组
```

仅禁止特定用户/组登陆

具体操作:

```
$ vi /etc/ssh/sshd_config
DenyUsers ramesh john jason #禁止登陆的用户
DenyGroups sysadmin dba #禁止登陆的用户组
```

更改sshd的默认端口

SSHD默认登陆端口为 `22` , 安全起见(防止被爆破), 最好改成一个乱七八糟的端口

```
$ vi /etc/ssh/sshd_config
Port 23333
```

更改登陆时限

默认的时间限制是2分钟, 如果2分钟内没有成功登陆, 服务器就会断开连接. 2分钟貌似有点长, 所以我们最好把他改小点:

```
$ vi /etc/ssh/sshd_config
LoginGraceTime 1m
```

更改监听的网卡

假设服务器有四个网卡, 每个网卡对应的IP地址分别是:

- eth0 – 192.168.10.200
- eth1 – 192.168.10.201
- eth2 – 192.168.10.202
- eth3 – 192.168.10.203

但是你只想在特定的网卡上监听服务, 那么你就可以在配置文件中写道:

```
$ vi /etc/ssh/sshd_config
ListenAddress 192.168.10.200 # 这是网卡0
ListenAddress 192.168.10.202 # 这是网卡2
```

不活动时断开连接

这个不活动, 指的是没有命令执行,无论命令成功获失败, 也就是说, 只要你在某一时间段内没有按下回车,且当前没有任务在运行, 那么服务器就会主动断开连接(把你踢出去).

如果是在bash里面,可以利用 `TMOUT` 这个变量.

在OpenSSH中, 可以这样修改:

```
$ vi /etc/ssh/sshd_config
ClientAliveInterval 600
ClientAliveCountMax 0 #从不检查
```

设置600内,如果无活动就断开连接.

Hack-32 PuTTY

这里是介绍windows上的ssh连接软件 PuTTY, 而且还是一些关于注册表乱七八糟的, 我就不翻译了, 因为没用.

(这个作者越来越不像话了嘿!)

我说点别的吧, ssh密钥登陆:

在这个ssh配置文件(`/etc/ssh/sshd_config`)中,更改如下地方:

```
PubkeyAuthentication yes
#AuthorizedKeysFile    %h/.ssh/authorized_keys # 这个是默认的,不用改

...
# Change to no to disable tunnelled clear text passwords
PasswordAuthentication no # 关闭密码登录, 只允许通过密钥登陆
```

然后在你本机生成一个rsa的密钥:

用它 -> `ssh-keygen`

一步一步生成, 有提示的. # 也可以一路回车 :)

然后将生成的公钥, 以 `.pub` 结尾的那个, 拷贝到服务器的 `~/.ssh` 目录下, 然后就可以直接登陆啦~

如果不拷贝的话,可以用命令 `ssh-copy-id user@host` 这个时候就需要输入一次密码.

同样很方便.

第三章到此结束咯, 是不是感觉没学到什么东西...(除了 `TMOUT` 这个变量)

第四章 - 日期设置

这一章介绍了5个关于日期设置的命令.

设置系统时间

`date` 这个命令是用来显示/设置时间的.

如果要设置本系统的时间, 命令如下:

```
$ date {mmddhhmiyyyy.ss}
```

其中:

- `mm` 几月, 两位
- `dd` 几号, 两位
- `hh` 几点, 两位
- `mi` 几分, 两位
- `yyyy` 哪一年, 四位
- `ss` 第几秒, 两位

例如, 设置日期为 1995年4月3日2点1分0秒:

```
$ date 040302011995.00 #需要root权限
```

如果只设置时间(不包括年月日):

```
$ date +%T -s "22:19:53"
$ date +%T%p -s "10:19:53PM"
```

设置硬件时间

硬件时间和系统时间有什么区别吗?

当然有了,硬件时间是主板上的时间,而系统时间则是系统里面的时间.

> 系统用两个时钟保存时间：硬件时钟和系统时钟。
>
> 硬件时钟(即实时时钟 RTC 或 CMOS 时钟)仅能保存：年、月、日、时、分、秒这些时间数值，无法保存时间标准(UTC 或 localtime)和是否使用夏令时调节。
>
> 系统时钟(即软件时间) 与硬件时间分别维护，保存了：时间、时区和夏令时设置。Linux 内核保存为自 UTC 时间 1970 年1月1日经过的秒数。系统启动之后，系统时钟与硬件时钟独立运行，Linux 通过时钟中断计数维护系统时钟。

via: wiki.archlinux.org

如果你有双系统的话,你就会发现,Linux上的时间和Windows上的时间相差8个小时,这是为什么呢?因为两种系统读时间的姿势不对,虽然他们都是从主板里面读取时间,但是Windows默认的读法是直接读,也就是说,主板里写的是啥,他就读成啥,但是Linux呢,Linux就会把主板里的时间换算成UTC,UTC是啥?国际时间,格林尼治时间,然后看一下我们在中国,+8区,再加8个小时,所以双系统的情况下就会相差8个小时了.

欲知详情,请看:linux系统时间和硬件时钟问题

几个命令:

- 显示硬件时间 `hwclock`
- 设置硬件时间为系统时间 `hwclock --systohc`

格式化日期

下面的例子是用不同的格式来显示当前日期:

```
➤ date
2016年 01月 04日 星期一 16:56:44 CST
➤ date --date='now'
2016年 01月 04日 星期一 16:56:55 CST
➤ date --date='tomorrow'
2016年 01月 05日 星期二 16:56:59 CST
➤ date --date='yestoday'
date: invalid date ‘yestoday’
➤ date --date='today'
2016年 01月 04日 星期一 16:57:11 CST
➤ date --date='1970-01-01 00:00:01 UTC +5 hours' +%s
18001
➤ date '+Current Date: %m/%d/%y%nCurrent Time:%H:%M:%S'
Current Date: 01/04/16
Current Time:16:57:25
➤ date +"%d-%m-%Y"
04-01-2016
➤ date +"%d/%m/%Y"
04/01/2016
➤ date +"%A,%B %d %Y"
星期一,一月 04 2016
➤
```

解释相关选项:

- `%D` 日期 (mm/dd/yy)
- `%d` 第几号 (01..31)
- `%m` 月份 (01..12)
- `%y` 年份的后两位 (00..99)
- `%a` 周几 (Sun..Sat)
- `%A` 周几 (Sunday..Saturday)
- `%b` 月份 (Jan..Dec)
- `%B` 月份 (January..December)

- `%H` 几点 (00..23)
- `%I` 几点 (01..12)
- `%Y` 年份 (1970...)

`date` 还有一个很有用的功能就是转换时间戳, 比如, 把现在的时间转换成Unix时间戳:

```
➤ date +%s
1451901927
```

这个时间戳, 就是从1970-1-1数过来的秒数.

显示过去的时间

这里没什么好说的.

```
$ date --date='3 seconds ago'
Thu Jan
1 08:27:00 PST 2009
$ date --date="1 day ago"
Wed Dec 31 08:27:13 PST 2008
$ date --date="1 days ago"
Wed Dec 31 08:27:18 PST 2008
$ date --date="1 month ago"
Mon Dec
1 08:27:23 PST 2008
$ date --date="1 year ago"
Tue Jan
1 08:27:28 PST 2008
$ date --date="yesterday"
Wed Dec 31 08:27:34 PST 2008
$ date --date="10 months 2 day ago"
Thu Feb 28 08:27:41 PST 2008
```

显示未来的时间

同样也没什么好说的.

```
$ date
Thu Jan
1 08:30:07 PST 2009
$ date --date='3 seconds'
Thu Jan
1 08:30:12 PST 2009
$ date --date='4 hours'
Thu Jan
1 12:30:17 PST 2009
$ date --date='tomorrow'
Fri Jan
2 08:30:25 PST 2009
$ date --date="1 day"
Fri Jan
2 08:30:31 PST 2009
$ date --date="1 days"
Fri Jan
2 08:30:38 PST 2009
$ date --date="2 days"
Sat Jan
3 08:30:43 PST 2009
$ date --date='1 month'
Sun Feb
1 08:30:48 PST 2009
$ date --date='1 week'
Thu Jan
8 08:30:53 PST 2009
$ date --date="2 months"
Sun Mar
1 08:30:58 PST 2009
$ date --date="2 years"
Sat Jan
1 08:31:03 PST 2011
$ date --date="next day"
```

```
Fri Jan
2 08:31:10 PST 2009
$ date --date="-1 days ago"
Fri Jan
2 08:31:15 PST 2009
$ date --date="this Wednesday"
Wed Jan
7 00:00:00 PST 2009
```

第五章 - PS* 介绍

这一章介绍了`PS1`, `PS2`, `PS3`, `PS4`, 以及`PROMPT_COMMAND`.

所谓的这些`PS`,实际上就是我们看到的提示符, 比如在终端上你看到的`name@host~ $`.

各位看官且跟我走~

PS1

`PS1` 就是你每次打开终端，首先显示的提示符，比如 `kity@cat ~$` .

在你的 `~/.bashrc` 中已经定义了默认的 `PS1` :

```
➤ cat .bashrc | grep 'PS1'
#    PS1='${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]\$ '
#    PS1='${debian_chroot:+($debian_chroot)}\u@\h:\w\$ '
    PS1="\[\e]0;${debian_chroot:+($debian_chroot)}\u@\h: \w\a\]$PS1"
➤
```

那你就会说，哎，为啥你的终端和我的不一样啊，你的怎么是个箭头呢?

那是因为我修改过呀:

```
if [ `whoami` == root ]; then
    PS1='\[\033[01;32m\]\u\[\033[00m\]:\[\033[01;34m\]\[\033[10;31m\]# \[\033[00;31m\]\[\033[00m\]'
else
    PS1='\[\033[00;31m\]➤ \[\033[00;31m\]\[\033[00m\]'
fi
```

是不是很low...

言归正传，解释一下上面的每个参数都代表什么:

- `\u` 代表用户名，取决于 `whoami`
- `\h` 主机名,取决于 `hostname`
- `\w` 当前目录的绝对地址，取决于 `pwd`

还有好多作者没有介绍的，我在此把他写的另一篇文章放上来:

看这里看这里!

算是个扩展阅读吧!

PS2

`PS2` 是当你敲命令换行的时候所提示你输入的字符，比如:

```
➤ echo "hi
> hello world!
> "
hi
hello world!

➤
```

在这里,' `>` '就是 `PS2` ，如果更改 `PS2` 的值之后:

```
➤ PS2='#'
➤ echo "hi
#hello world!
#"
hi
hello world!

➤
```

就会这样了~

你可以按照你的喜好改成别的字符或是字符串.

PS3

`PS3` 这个变量只存在于一个地方，即 `select` 选择的时候，提示输入选择的内容:

```
ramesh@dev-db ~> cat ps3.sh
select i in mon tue wed exit
do
    case $i in
        mon) echo "Monday";;
        tue) echo "Tuesday";;
        wed) echo "Wednesday";;
        exit) exit;;
    esac
done
ramesh@dev-db ~> ./ps3.sh
1) mon
2) tue
3) wed
4) exit
#? 1
Monday
#? 4
```

默认的提示符是 `#?` .

然后我们更改一下:

```
ramesh@dev-db ~> cat ps3.sh
PS3="Select a day (1-4): "
select i in mon tue wed exit
do
    case $i in
        mon) echo "Monday";;
        tue) echo "Tuesday";;
        wed) echo "Wednesday";;
        exit) exit;;
    esac
done
ramesh@dev-db ~> ./ps3.sh
1) mon
2) tue
3) wed
4) exit
Select a day (1-4): 1
Monday
Select a day (1-4): 4
```

如果不在文件中更改的话,也可以 `export` 一下(还是用第一个未更改过的脚本):

```
➤ cat ps3.sh
select i in mon tue wed exit
do
    case $i in
        mon) echo "Monday";;
        tue) echo "Tuesday";;
        wed) echo "Wednesday";;
        exit) exit;;
    esac
done
➤ export PS3='yoooo--> '
➤ bash ps3.sh
1) mon
2) tue
3) wed
4) exit
yoooo--> 2
Tuesday
yoooo--> 4
➤
```

PS4

这可不是游戏机哦~

`PS4` 这个变量存在于调试过程中，也就是开起了 `set -x` 之后：

```
➤ cat ps4.sh
set -x
echo "PS4 demo script"
ls -l /etc/ | wc -l
du -sh .
➤
➤ bash ps4.sh
+ echo 'PS4 demo script'
PS4 demo script
+ ls -l /etc/
+ wc -l
285
+ du -sh .
4.9M    .
```

默认的 `PS4` 是一个加号.

下面更改一下

```
➤ export PS4='$0.$LINENO+ '
➤ bash ps4.sh
ps4.sh.3+ echo 'PS4 demo script'
PS4 demo script
ps4.sh.4+ ls -l /etc/
ps4.sh.4+ wc -l
285
ps4.sh.5+ du -sh .
4.9M    .
➤
```

其中 `$0` 是脚本名字，`$LINENO` 是命令所在的行号.

PROMPT_COMMAND

`PROMPT_COMMAND` 指的是当命令运行结束后所输出的字符.

比如:

```
➤ echo $PROMPT_COMMAND
echo -ne "\033]0;${USER}@${HOSTNAME}: ${PWD/$HOME/~}\007"
➤ echo -ne "\033]0;${USER}@${HOSTNAME}: ${PWD/$HOME/~}\007"
➤
➤
```

这个是啥也没有的输出...

咱改一改:

```
➤ PROMPT_COMMAND='echo "Hello world!"'
Hello world!
➤ whoami
mr
Hello world!
➤ pwd
/home/mr/test
Hello world!
➤ date
2016年 01月 04日 星期一 22:17:24 CST
Hello world!
➤
```

看到了? 把 `PROMPT_COMMAND` 改成"Hello world!"之后, 每次命令结束都会再输出一个"Hello world!", 我们可以在这里做一点小动作:

作者把它改成了时间:

`export PROMPT_COMMAND="date +%H:%M:%S"`

```
➤ pwd
/home/mr/test
22:21:34
➤ whoami
mr
22:21:36
➤
```

我觉着没卵用，倒不如这样好玩:

先自定义一个函数:

```
function ttt() { [[ $? -eq 0 ]] && echo -n yes || echo -n no; }
```

然后:

```
export PROMPT_COMMAND="ttt"
```

这样每次命令完成都有反馈啦~

(虽然也没什么卵用...

自定义PS1

1.在提示符里输出用户名,主机名, 当前目录:

```
export PS1="\u@\h \W> "
```

其中:

- `\W` 是当前目录的 basename, 也就是目录名, 不带绝对路径的.

其他的在之前已经说过, 不再重复.

2.在提示符里输出当前时间:

```
export PS1="\u@\h [\$(date +%H:%M:%S)]> "
```

`PS1` 中可以带命令, 正如上面的例子, 输出时便会附带当前时间.

上面的 `$(date +%H:%M:%S)` 可以替换为: `\t`

或者用 `\@` 输出当前的小时和分钟.

3.显示任何命令:

其实,这个说法并不是很准确, 因为自定义的命令不会运行, 除非像上一篇那样, 在外部定义了一个自己的命令, 否则, 只会输出第一次命令运行得到的结果, 听起来可能有点啰嗦, 你自己动手试一下就好了.

这里再多说几个:

......

妈蛋, 原作者说的都是些啥啊, 越来越水了... 没用的就不翻译了.

说点自己的经验:

1. 这些变量是类似于一个子shell运行的, 外部命令不会对内部产生影响
2. 变量可以是一条命令, 但是这条命令必须是系统自带的, 自己写的函数不会起作

用.
3. 自己在外部写的函数会在里面被引用, 不知道是替换还是什么, 总之能够运行.
4. 不动手试一下你就不知道我说的是什么...

4.用内部已有的代码自定义PS1

如果你看过之前的那篇文章(Hack-38 的扩展阅读部分), 这里的东西就当是复习了.

先列举一下那些内部代码, 类似于 `\n` 代表换行符一样:

- `\a` 响铃
- `\d` 日期
- `\D{format}` 自定义的日期
- `\e` 逃逸字符
- `\h` 主机名(前半部分)
- `\H` 主机名(完整的)
- `\j` 当前shell下的后台job数量, 相当于 `jobs`
- `\l` shell终端的basename... (这个都给定义了...
- `|n` 换行
- `\r` 你知道 `\r` 和 `\n` 的区别嘛 (这个是回车, 上面的是换行哦~)
- `\s` shell的名字
- `\t` 24小时制的时间 - HH:MM:SS
- `\T` 12小时制的时间 - HH:MM:SS
- `\@` 12小时制带上下午的时间 - am/pm (真啰嗦啊...
- `\A` 24小时制的时间 - HH:MM
- `\u` 当前用户名
- `\v` 当前Bash的版本号 (我真是醉了...
- `\V` Bash的发布版本号 4.3.42 (可以理解为较长的那个
- `\w` 当前目录(绝对路径)
- `\W` 当前目录的短名字 (可以理解为目录名
- `\!` 这条命令在历史记录中的编号
- `\#` 这条命令在当前shell中的编号
- `\$` 如果 `$UID -eq 0` 那么这个就输出 `#` , 否则输出 `$`
- `\nnn` nnn表示一个八进制的数字, 整体就表示这个八进制的字符
- `\\` 一个反斜杠
- `\[` 转义开中括号
- `\]` 转义闭中括号

5.在 `PS1` 中运行自定义function

哈哈，我翻译 `PROMPT_COMMAND` 那一部分的时候还没看到这里呢，所以不算剧透哦，因为我也不知道作者写了这一部分，而且上面的在 `PS*` 变量中自定义功能可是我举一反三得来的哦~

所以，这里作者说的是定义了一个外部 `function`，然后在从 `PS1` 里面调用.

这样你的选择就多了去了，随你想干什么，bash 都满足你哦~ 哈哈~

6.在 `PS1` 中运行脚本

在 `PS1` 变量中既然可以运行命令，那么同样也可以运行脚本.

假如你在 `~/bin/totalfilesize.sh` 中存放着一个内容如下的脚本:

```
#!/bin/bash
for filesize in $(ls -l . | grep "^-" | awk '{print $5}')
do
let totalsize=$totalsize+$filesize
done
echo -n "$totalsize"
```

正如你所看到的，这个脚本的作用是计算当前目录下文件的大小.

然后我们将 `PS1` 的值改掉: `export PS1="\u@\h [\$(totalfilesize.sh) bytes]> "`

那么每当你敲回车的时候都会看到当前目录下的文件总大小:

```
ramesh@dev-db [534 bytes]> cd /etc/mail
ramesh@dev-db [167997 bytes]>
```

PS:可以把脚本内容改成:

```
ls -l | awk '/^-/ { sum+=$5 } END { printf sum }'
```

这样会简练一些.

给点颜色给PS1

这里介绍了一堆关于彩色显示的代码...

1.字体颜色

用下面的代码可以让 `PS1` 变成蓝色:

```
export PS1="\e[0;34m\u@\h \w> \e[m "
```

知道这是为什么嘛?

因为我们在中间嵌入了颜色代码: `\e[0;34m` 以及关闭颜色的代码: `\e[m`

下面的命令会让颜色变得更亮, 还带有加粗效果:

```
export PS1="\e[1;34m\u@\h \w> \e[m "
```

慢慢介绍里面的东西都是什么意思:

- `\e[` 颜色开启
- `x;ym` 颜色的类型: x,y自己定义
- `\e[m` 关闭颜色

下面是一张颜色代码表:

颜色	代码
黑色	0;30
蓝色	0;34
绿色	0;32
青色	0;36
红色	0;31
紫色	0;35
棕色	0;33

如果把上面代码中的 `0` 替换成 `1` 就会使颜色加深, 字体加粗.

2.字体背景色

我们不仅可以更改显示的字体的颜色, 我们还可以改变字体的背景颜色:

```
export PS1="\e[47m\u@\h \w> \e[m "
```

上面的命令使我们得到了一个浅灰色的背景.

3.组合

没错, 我们当然可以把上两种结果组合起来, 实现任何你想要的结果:

```
export PS1="\e[0;34m\e[47m\u@\h \w> \e[m "
`
```

比如利用上面的代码就可以得到一个字体颜色为蓝色, 字体背景色为浅灰的效果.

下面是我自己截的图, 不要看花眼...

```
~> export PS1="\e[0;34m\e[40m\w> \e[m "
~> export PS1="\e[0;34m\e[41m\w> \e[m "
~> export PS1="\e[0;34m\e[42m\w> \e[m "
~> export PS1="\e[0;34m\e[43m\w> \e[m "
~> export PS1="\e[0;34m\e[44m\w> \e[m "
~> export PS1="\e[0;34m\e[45m\w> \e[m "
~> export PS1="\e[0;34m\e[46m\w> \e[m "
~> export PS1="\e[0;34m\e[47m\w> \e[m "
~> export PS1="\e[0;34m\e[48m\w> \e[m "
~>
```

每一行命令都对应它下一行开头的颜色.

这里还是希望你亲自实验一下, 而不是看一遍...

4. `tput` 命令

说实话, 我对这里也是第一次见, 长知识了.

例子:

```
$ export PS1="\[$(tput bold)$(tput setb 4)$(tput setaf
7)\]\u@\h:\w $ \[$(tput sgr0)\]"
```

其中:

- `tput setb [1-7]` - 设置背景色
- `tput setf [1-7]` - 设置前景色
- `tput setab [1-7]` - 用ANSI escape设置背景色(我查了,但不懂...)
- `tput setaf [1-7]` - 用ANSI escape设置前景色

前两个好使,就算用 `echo` 输出也是很漂亮的.

还有一些设置字体的选项:

- `tput bold` 粗体
- `tput dim` 亮度减半
- `tput smul` 开启下划线
- `tput rmul` 关闭下划线
- `tput rev` 颜色反转,类似于高对比度那种
- `tput smso` 开启高对比度模式
- `tput rmso` 关闭高对比度模式
- `tput sgr0` 关闭所有特效

颜色代码:

- `0` – 黑色
- `1` – 红色
- `2` – 绿色
- `3` – 黄色
- `4` – 蓝色
- `5` – 洋红
- `6` – 青色
- `7` - 白色

扩展阅读

9 UNIX / Linux tput Examples: Control Your Terminal Color and Cursor

第六章 - 压缩和打包

这一章介绍了压缩打包之类的命令技巧.

你会在传输大文件以及"脱裤"的时候用到的 :)

Zip 命令基础

`zip` 打包，是最好用的了，不仅命令简单，压缩率也不低哦~

基础语法：

```
zip {.zip file-name} {file-names}
```

如果要打包的文件/文件夹下还有文件夹怎么办？递归啊！

```
zip -r var-log-dir.zip /var/log/ # -r 开关， 开启递归选项
```

那解压呢？

也是相当的简单..

```
unzip zipfile.zip
```

直接用 `unzip` 就好了！

那如果只是想看看zip包里面的内容而不解压呢？

加一个 `-l` 的参数就好了！

```
unzip -l zipfile.zip
```

Zip 命令进阶

`zip` 一共有10个压缩等级:

- `0` 最低的等级, 只是打包而不做任何压缩
- `1` 压缩率很低, 但是速度很快
- `6` 默认压缩等级
- `9` 最高等级的压缩!速度很慢, 但压缩率是最高的. 作者认为, 如果不是压缩一个很大的文件, 用最高等级是最好的选择. (不敢苟同啊... 我一直认为默认的是最优的 :D

下面看一下各个压缩等级有什么不同:

这是要操作的文件:

```
➤ seq 9999999 > big_file
➤ l big_file
-rw-rw-r-- 1 mr mr 76M  1月  5 17:04 big_file
➤
```

然后看一下压缩效果:

```
➤ zip 0.zip big_file
  adding: big_file (deflated 73%)
➤ zip -0 0.zip big_file
updating: big_file (stored 0%)
➤ zip 6.zip big_file
  adding: big_file (deflated 73%)
➤ zip 9.zip big_file
  adding: big_file (deflated 73%)
➤
➤ ls -l *.zip
-rw-rw-r-- 1 mr mr 78889054  1月  5 17:05 0.zip
-rw-rw-r-- 1 mr mr 21230796  1月  5 17:05 6.zip
-rw-rw-r-- 1 mr mr 21230796  1月  5 17:05 9.zip
```

验证一个压缩包

有时候我们想验证一个压缩包里的文件是否完整，以及，这是不是我们想要的压缩包，那你就会用到 `-t` 这个参数：

```
➤ unzip -t 9.zip
Archive:  9.zip
    testing: big_file                 OK
No errors detected in compressed data of 9.zip.
```

给压缩包加个锁

我们知道 `RAR` 可以添加密码，其实， `ZIP` 也可以的.

命令:

```
zip -P mysecurepwd var-log-protected.zip /var/log/*
```

如果你觉得这样明文记录密码不好的话，还可以交互式的输入密码:

```
zip -e var-log-protected.zip /var/log/*
Enter password:
Verify password:
updating: var/log/acpid (deflated 81%)
updating: var/log/anaconda.log (deflated 79%)
```

当然，输入的密码是不可见的.

当你解压这个带锁的压缩包的时候就会要求你输入密码:

```
unzip
Archive:
var-log-protected.zip
var-log-protected.zip
[var-log-protected.zip] var/log/acpid password:
```

Tar 命令

除了 `zip` 这个简单实用的压缩工具以外, Linux上应用最广泛的还是这个 `tar` .

命令语法:

```
tar [options] [tar-archive-name] [other-file-names]
```

常用的选项有这么几个:

- `c` 创建一个归档文件
- `v` 显示详细过程
- `f` 后接要创建的文件名
- `z` 压缩(.gz)
- `j` 压缩(.bz2)
- `x` 解压
- `t` 测试(也就是查看里面都有啥, 并不解压)

1.简单的创建一个归档文件(不压缩)

```
tar cvf /tmp/my_home_directory.tar /home/jsmith
```

这里的 `option` 前面加不加 `-` (连字符)都行, 推荐还是加上, 更美观, 也符合基本认知.

2.查看某个压缩包里的内容(列出文件)

```
tar tvf /tmp/my_home_directory.tar
```

3.解压某个压缩包(提取文件)

```
tar xvf /tmp/my_home_directory.tar
```

4.解压文件到特定的目录

```
tar xvfz /tmp/my_home_directory.tar.gz -C /tmp/some_dir
```

扩展阅读

[Additional Tar Examples]

Tar 压缩

`tar` 可以和 `gzip` , `bzip2` 组合使用, 也就是边打包,边压缩.

上一篇说了打包, 这一篇就介绍压缩.

其中压缩有两个软件, 一个是 `gzip` , 创建的是 `.gz` 的文件, 另一个是 `bzip2` ,创建的是 `.bz2` 的文件, 两者有什么区别呢? 可以简单地理解为,在相同的条件下, `.bz2` 压缩率更高, 更能节省空间.

gzip压缩

```
$ tar cvfz /tmp/my_home_directory.tar.gz /home/jsmith #压缩
$ tar xvfz /tmp/my_home_directory.tar.gz #解压
$ tar tvfz /tmp/my_home_directory.tar.gz
```

还记得这些参数都是什么意思吗?

不记得就往回翻翻看~

bzip2压缩

```
$ tar cvfj /tmp/my_home_directory.tar.bz2 /home/jsmith #压缩
$ tar xvfj /tmp/my_home_directory.tar.bz2 #解压
$ tar tvfj /tmp/my_home_directory.tar.bz2
```

看出两者有何区别了么?

Bz* 命令

`bzip2` 是用来压缩和解压文件的一个命令, 它最大的优点在于超高的压缩效率.

`bzip2` VS `gzip`

- `bzip2` 压缩率更高
- `gzip` 速度更快
- 在高压缩率下 `bzip2` 的速度更块一些(相比 `gzip`)

压缩某个文件

```
➤ ls -l catshadow
-rwxrwxr-x 1 mr mr 8.7K 11月 11 15:49 catshadow*
➤ bzip2 catshadow
➤ ls -l catshadow.bz2
-rwxrwxr-x 1 mr mr 2.8K 11月 11 15:49 catshadow.bz2*
```

然后你就会发现原来的文件不见惹...

在压缩文件(.bz2)里面搜索 (bzgrep)

如果你压缩了某个日志文件, 想搜索里面的某条信息, 那你是不是应该先解压, 再用 `grep` 搜索?

然而,这条命令就把两者组合起来了:

```
bzgrep grep-options -e pattern filename
```

我觉着这是不符合Unix哲学的, 把自己的事情做好就行了, 可是话又说回来, 你怎么知道我想做什么事呢?

```
➤ seq 99999 > numbers
➤ l numbers
-rw-rw-r-- 1 mr mr 576K  1月  5 20:28 numbers
➤ bzip2 numbers
➤ bzgrep 233 numbers.bz2
233
1233
2233
2330
2331
...
...
```

直接在文件里面搜索.

查看压缩文件的内容(不解压)

同样, 我们不解压也可以直接查看压缩文件的内容:

```
bzcat numbers.bz2
```

或者:

```
bzless numbers.bz2
```

或者:

```
bzmore numbers.bz2
```

bz找不同

还有 `bzcmp` 和 `bzdiff` 这两个命令, 是用来找不同的.

```
$ cmp System.txt.001 System.txt.002
System.txt.001 System.txt.002 differ: byte 20, line 2
$ bzcmp System.txt.001.bz2 System.txt.002.bz2
- /tmp/bzdiff.csgqG32029 differ: byte 20, line 2
```

```
$ bzdiff System.txt.001.bz2 System.txt.002.bz2
2c2
< 0: ERR: Mon Sep 27 12:19:34 2010: gs(11153/1105824064):
[chk_sqlcode.scp:92]: Database: ORA-01654: unable to
extend index OPC_OP.OPCX
_ANNO_NUM by 64 in tablespace OPC_INDEX1
---
> 0: ERR: Wed Sep 22 09:59:42 2010:
gs(11153/47752677794640): [chk_sqlcode.scp:92]: Database:
ORA-01653: unable to extend table OPC_OP.
OPC_HIST_MESSAGES by 64 in tablespace OPC_6
4,5c4
< Retry. (OpC51-22)
< Database: ORA-01654: unable to extend index
OPC_OP.OPCX_ANNO_NUM by 64 in tablespace OPC_INDEX1
---
> 0: ERR: Wed Sep 22 09:59:47 2010:
gs(11153/47752677794640): [chk_sqlcode.scp:92]: Database:
ORA-01653: unable to extend table OPC_OP.
OPC_HIST_MESSAGES by 64 in tablespace OPC_6
```

这一部分我介绍的有点少，是因为我本人用bz系列不是很多，为了追求速度，都是用 `tar` 的，而且日常生活中也遇不到对比两个压缩文件内容的情况...

Cpio 命令

这个 `cpio` 我是第一次听说, 如有不妥的地方还请大家指正.

`cpio` 命令是用来处理归档文件的, 这里的归档文件包括 `.cpio` , `.tar`

> cpio stands for “copy in, copy out”.

说的很明白, 复制进来, 复制出去. 果真Linux的软件命名都是根据内容来的, 直观易懂.

它可以干三种事:

1. 把文件复制到某个归档文件中
2. 从某个归档文件中提取文件
3.

`cpio` 从标准输入中读取文件列表, 创建一个归档文件后把这些文件都输入到里面, 最后再输出到标准输出中(或者重定向).

创建**cpio**归档

```
➤ cd test/
➤ ls
cal_random.sh   catshadow.c    numbers.bz2   vpnn
catshadow.bz2   helloword.py   test.php
➤ ls | cpio -ov > test.cpio   # o-创建归档文件
cal_random.sh
catshadow.bz2
catshadow.c
helloword.py
numbers.bz2
test.php
vpnn
248 blocks
➤ ls -l test.cpio
-rw-rw-r-- 1 mr mr 124K  1月   5 20:46 test.cpio
```

正如你所看到的,把 `ls` 列出的文件通过管道传递给 `cpio` 后, `cpio` 将他们压缩,然后我们再通过重定向,导入到了 `test.cpio` 文件中.

提取cpio中的文件

接着上一个目录中的内容,我们新建一个目录,把文件提取出来:

```
➤ mkdir cpio
mkdir: created directory ‘cpio’
➤ cd cpio/
➤ ls
➤ cpio -idv < ../test.cpio # i-从归档文件中提取
cal_random.sh
catshadow.bz2
catshadow.c
helloword.py
numbers.bz2
test.cpio
test.php
vpnn
494 blocks
➤ ls
cal_random.sh  catshadow.c   numbers.bz2  test.php
catshadow.bz2  helloword.py  test.cpio    vpnn
➤
```

看到了么, `cpio` 从标准输入中读取了归档文件,然后把里面的文件提取了出来.

归档特定的文件

```
➤ find . -iname *.c -print | cpio -ov >/tmp/c_files.cpio
./catshadow.c
1 block
➤
```

这里没啥好说的,就是利用了 `find` 而已.

用cpio创建tar文件

我们可以用 `cpio` 创建一个 `.tar` 类型的文件:

```
ls | cpio -ov -H tar -F sample.tar
```

殊途同归.

怎样提取呢?

```
cpio -idv -F sample.tar
```

用上面这个.

我们可以看到, 除了利用重定向, 我们还可以用 `-F` 的参数来定义所要操作的文件.

还有, 不解压查看 `tar` 文件里面的文件名:

```
cpio -it -F sample.tar
```

作者还列举了几个不常用的:

1.将符号链接所指向的内容打包:

```
ls | cpio -oLv >/tmp/test.cpio
```

2.保留文件的修改时间

```
ls | cpio -omv >/tmp/test.cpio
```

3.拷贝文件夹

```
$ mkdir /mnt/out
$ cd objects
$ find . -depth | cpio -pmdv /mnt/out
```

个人感觉 `cpio` 像是一个文件流操作器, 压缩也好, 解压也好, 复制也好, 都是以一种数据流的形式进行操作.

扩展阅读

- Additional cpio Examples
- 讨论

第七章 - 历史命令

这一部分介绍了三个关于 `history` 的技巧, 或是介绍.

用好了历史命令, 会让你的工作效率大大增加!

History 命令

> 你敲的每一个命令都会被忠实地记录下来 --- 我.

上面说的也不完全对啦, 因为这个"忠实"会被各种方法破解...

首先你要有一个历史记录的概念, 然后你要知道这些记录存放到哪儿, 最后你要知道这些记录可以被删除, 修改.

搜索历史命令

快捷键 `Ctrl + r`

非常建议你使用这个命令, 因为当你曾经输过一个很长的命令之后, 当你再次想输入这个命令的时候, 你就可以按下这个快捷键, 然后键入那条长命令的关键词, 然后就会显示出含有那个关键词的命令, 每次按下这个键都会再往上搜一个.

(还有一些其他的快捷键, 在我的博客上: HERE)

重复上一次的命令

1. 向上的方向键
2. 两个叹号: `!!`
3. 还有这个: `!-1`
4. 快捷键 `Ctrl+p`

最好用的还是方向键, 不是么~

从历史记录中执行某个命令

还是沿袭上一个中的 `!-n` 模式, 其中 `n` 是一个编号.

```
# history | more
1 service network restart
2 exit
3 id
4 cat /etc/redhat-release
# !4
cat /etc/redhat-release
```

作者给出的例子中, 执行了编号为 `4` 的命令.

执行曾经的命令中特定开头的

假设你的部分历史命令如下:

```
 1719  find . -type f
 1720  find . -type f | cpio -o > test.cpio
 1721  find .  | cpio -o > test.cpio
 1722  ls
 1723  l
 1724  du -h
```

那么, 怎样重复执行1721条呢? 除了利用 `!-1721` 这么麻烦的方法, 我们还可以用 `!f` 这样的姿势.

因为开头的 `f` 是离着最后一条命令最近的, 所以 `!f` 就执行了它.

清空历史记录

```
 history -c
```

从历史记录中截取参数

在下面的例子中, `!!:$` 等价于上一条命令的最后一个参数:

```
# ls anaconda-ks.cfg
anaconda-ks.cfg
# vi !!:$
vi anaconda-ks.cfg
```

下面的例子中，`!^` 则等价于上一条命令的第一个参数:

```
# cp anaconda-ks.cfg anaconda-ks.cfg.bak
anaconda-ks.cfg
# vi !^
vi anaconda-ks.cfg
```

从特定的历史记录中截取特定的参数

下面的例子中，`!cp:2` 等价于当前历史记录中，最后一个以 `cp` 开头的命令的第二个参数；`!cp:$` 则等价于当前历史记录中最后一个以 `cp` 开头的命令的最后一个参数.

是不是有点啰嗦...

```
# cp ~/longname.txt /really/a/very/long/path/long-
filename.txt
... ... ...
# ls -l !cp:2
ls -l /really/a/very/long/path/long-filename.txt
# ls -l !cp:$
ls -l /really/a/very/long/path/long-filename.txt
```

和历史命令相关的变量

在历史记录中显示时间

我们可以用 `HISTTIMEFORMAT` 这个变量来定义显示历史记录时的时间参数:

```
➤ export HISTTIMEFORMAT='%F %T '
➤ history
... ... ...
 1740  2016-01-05 22:45:28 man history
 1741  2016-01-05 22:45:33 export HISTTIMEFORMAT='%F %T'
 1742  2016-01-05 22:45:34 history
 1743  2016-01-05 22:45:38 export HISTTIMEFORMAT='%F %T '
 1744  2016-01-05 22:45:40 history
➤
```

你也可以用下面的别名来定义显示历史命令的数量:

```
alias h1='history 10'
alias h2='history 20'
alias h3='history 30'
```

改变历史记录的大小限制.

在 `!/.bashrc` 中有这两个变量控制者bash储存的历史命令的数量:

```
# vi ~/.bash_profile
HISTSIZE=450
HISTFILESIZE=450
```

他们的含义就跟他们的名字一样.

用 `HISTFILE` 改变存放历史命令的文件.

```
# vi ~/.bash_profile
HISTFILE=/root/.commandline_warrior
```

(都是些没有太多用处的东西, 尽管可定制化水平很高...

用 `HISTCONTROL` 来删除重复的历史记录

下面的例子中, 有三个 `pwd` 命令, 那么在 `history` 中就会显示三次 `pwd` , 有点不那么人性化.

```
# pwd
/
# pwd
/
# pwd
/
# history | tail -4
44 pwd
45 pwd
46 pwd
47 history | tail -4
```

所以我们可以这样修改:

```
export HISTCONTROL=ignoredups
```

然后就不会出现相邻的重复记录了~

在历史命令中去重

如果你还嫌弃历史记录中的那些不相邻的重复记录, 你可以用这个:

```
export HISTCONTROL=erasedups
```

不记录某些命令

(这个一般是默认的)

```
export HISTCONTROL=ignorespace
```

这样历史记录中就不会储存以空格开头的命令了.

禁止记录历史命令

如果你不想记录任何历史命令的话, 可以这样做:

```
export HISTSIZE=0
```

把历史命令记录的大小设置为**0**

(还可以把历史记录的保存文件改成 `/dev/null` , 殊途同归 ~)

不记录某些命令II

```
export HISTIGNORE="pwd:ls:ls -ltr:"
```

这种方法的效果是, 历史记录不会记录这几个命令.

扩展阅读

15个例子

忽然想到, 这些扩展阅读也都是英文的啊.... 不知道各位看官有没有兴趣看呢?

History 扩展

用这个功能可以选择特定的历史记录, 不论是修改还是立即执行, 都可以完成.

这个扩展以 `!` (叹号)开头.

- `!!` 重复上一条命令
- `!10` 重复历史记录中第10条命令
- `!-2` 重复历史记录中倒数第二条命令
- `!string` 重复历史记录中最后一条以 `string` 开头的命令
- `!?string` 重复历史记录中最后一条包含 `string` 的命令
- `^str1^str2^` 把上一条命令中的 `str1` 替换成 `str2` , 然后再执行
- `!!:$` 扩展成上一条命令的最后一个参数 (完全可以用 `Alt+.` 快捷键啊!)
- `!string:n` 扩展成最后一条以 `string` 开头的命令的第 `n` 个参数....

`!?string` 栗子

假设你曾经执行过这样一条命令:

```
$ /usr/local/apache2/bin/apachectl restart
```

然后过了一会儿你想重复这个命令, 然后你这样做:

```
$ !apache
-bash: !apache: event not found
```

哔~ 找不到!

当然, 因为曾经的那条命令并不是以 `apache` 开头的, 但是:

```
$ !?apache
/usr/local/apache2/bin/apachectl restart
```

这样就会执行, 因为这条命令包含了 `apache` 这个字.

`^str1^str2^` 栗子

作者给的例子太牵强，我给个实用的：

```
➤ cat cal_random.sh
#!/bin/bash
echo hi
test(){
    for i in {1..9};do
        echo $i;
    done
}
... ... ...
exit 0
➤ ^cat^less
less cal_random.sh
➤
```

哈，这个也不实用，因为要达到这样的目的有好多方法，而替换这一种实在太麻烦了.

`!!:$` 栗子

废话不说，上代码 ~!

```
➤ cp cal_random.sh cal_random.sh.bak
➤ head -n 2 !!:$
head -n 2 cal_random.sh.bak
#!/bin/bash
echo hi
➤
```

在这个例子中， `!!:$` 被扩展成了上一条命令的最后一个参数，可以理解为正则的 `$` 表示最后一样.

`!string:n` 栗子

曾经执行了这样一条命令

```
cat file1 file2 file3
```

然后你想截取其中的第二个参数,

那么你就可以这样(确保最后一条以cat开头的命令是它):

```
ls !cat:2
```

上面这些东西用好了真的很方便! 尽管我一直用快捷键...

第八章 - 系统任务管理

这里有关于磁盘管理的, 用户管理的, 远程登陆的, 最后还介绍了Iptables...

可以理解为, 乱七八糟...

希望会对新手和老司机都有帮助 :)

~~翻译进展到一半了, 进度还是蛮快的, 其实现在想想, 当初的动机也不再那么纯洁了, 我也是想以后找工作的时候在简历上画一笔...毕竟都快毕不了业了...~~

也希望会对我有帮助 :P

Fdisk 命令

讲真, 这种东西挺危险的, 如果你不知道你在做什么, 请不要随意使用这个工具, 如果出了差错, 那你近几天的搜索关键词将会变成"数据恢复 Linux", 这是一个悲伤的故事...

所以个人建议新手还是不要实际操作, 知道个大概意思就可以, 当然, 你完全可以新划分一块分区进行操作, 或者边看鸟哥的书边操作, 毕竟这里说的都很肤浅, 三思而后行, 特别是对重要数据进行操作.

基本命令

`fdisk` 的操作键:

- `n` 新建一个分区
- `d` 删除一个分钱
- `p` 打印当前的分区表
- `w` 把当前所有操作写入分区表, 也就是保存. (三思啊主公!)
- `q` 退出

新建一个分区

下面的例子新建了一个 `/dev/sda1` 分区:

```
# fdisk /dev/sda
Command (m for help): p
Disk /dev/sda: 287.0 GB, 287005343744 bytes
255 heads, 63 sectors/track, 34893 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

Device   Boot   Start   End   Blocks   Id   System

Command (m for help): n
Command action
    e   extended
    p   primary partition (1-4)
p

Partition number (1-4): 1
First cylinder (1-34893, default 1):
Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-34893,
default 34893):
Using default value 34893

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```

不得不说, 这个作者越来越水了,这种分区的东西一年半载根本用不上一次, 介绍这么点东西还不如不说, 这里面水多深, 又是分区表格式, 又是磁盘格式, 这里面的区别, 道道, 一天半会儿根本学不来...

建议大家先不要触及这方面的东西, 要不就先看看操作系统, 了解一下磁盘分区, 了解一下储存方式以及Linux是怎么组织文件的, 要是什么都不懂, 还是离分区远远的吧, 数据弄丢了可真不是一件小事儿.(用虚拟机的就不要管我了...)

另外, 还有一些工具是用来恢复数据的, 不懂的千万要查文档, 否则你怎么死的都不知道,

Mke2fsk 命令

光分区了还不算, 还要给分的区建立一种格式, 让磁盘有组织有纪律的存放文件.

`mke2fsk` 就是这样一个格式化分区的工具.

回到之前创建的新分区(并不是让你回去创建分区啊! 很危险的! 看看就好了, 不要真把硬盘搞坏了), 查看一下分区的信息:

```
# tune2fs -l /dev/sda1
tune2fs 1.35 (28-Feb-2004)
tune2fs: Bad magic number in super-block while trying to open /d
ev/sda1
Couldn't find valid filesystem superblock.
```

看吧, 找不到 `superblock` , 话说这个 `superblock` 是个什么?

> A superblock is a record of the characteristics of a filesystem, including its size, the block size, the empty and the filled blocks and their respective counts, the size and location of the inode tables, the disk block map and usage information, and the size of the block groups.

可以简单地理解为一张大饼表, 上面记录了各种文件的信息, 系统的信息, 谁放在哪儿了, 哪里还有空位子, 等等等等...

然后我们开始格式化这个分区:

```
# mke2fs /dev/sda1
```

说道格式化分区, `Gparted` 实在是一个不可多得的好工具.

```
# mke2fs -m 0 -b 4096 /dev/sda1
mke2fs 1.35 (28-Feb-2004)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
205344 inodes, 70069497 blocks
0 blocks (0.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=71303168
2139 block groups
32768 blocks per group, 32768 fragments per group
96 inodes per group
Superblock backups stored on blocks:
32768, 98304, 163840, 229376, 294912, 819200, 884736,
1605632, 2654208, 4096000, 7962624, 11239424, 20480000,
23887872
Writing inode tables: done
Writing superblocks and filesystem accounting information:
done
This filesystem will be automatically checked every 32
mounts or 180 days, whichever comes first. Use tune2fs -c
or -i to override.
```

这里来一点注释:

- `-m 0` 设置保留块的百分比为0, 默认是5%. 保留的地方是给root的.
- `-b 4096` 设置每一块多少比特, 可用的有1024, 2048 和 4096.

上面的过程, 创建了一个 `ext2` 的分区.

你可以用下面的两条命令来创建 `ext3` 的分区:

```
# mkfs.ext3 /dev/sda1
# mke2fs –j /dev/sda1
```

当然, 还是推荐使用 `Gparted` , 图形化, 更直观.

挂载一个分区

新建并格式化某个分区之后, 你需要把他挂载到主机上.(一般都是自动挂载的... 所以作者在这儿也是给101充数.)

1.挑个地儿:

新建一个文件夹, 或者选择一个已有的空文件夹都好:

```
mkdir /home/data
```

2.把分区挂载到刚才的地方:

```
mount /dev/sda1 /home/data
```

3.卸载:

```
umount /dev/sda1
```

都是些很简单的命令.

如果想自动挂载这个分区, 可以写入到 `/etc/fstab` 中:

```
/dev/sda1 /home/database ext3 defaults 0 2
```

查看分区信息

这里也只是一个命令, 就是输出的有点多.

`tune2fs` ,这个在之前有用到过, 是用来查看分区信息的.

```
# tune2fs -l /dev/sda1
tune2fs 1.35 (28-Feb-2004)
Filesystem volume name: /home/database
Last mounted on: <not available>
Filesystem UUID: f1234556-e123-1234-abcd-bbbbaaaaae11
Filesystem magic number: 0xEF44
Filesystem revision #: 1 (dynamic)
Filesystem features:
sparse_super resize_inode filetype
Default mount options: (none)
Filesystem state: not clean
Errors behavior: Continue
Filesystem OS type: Linux
Inode count: 1094912
Block count: 140138994
Reserved block count: 0
Free blocks: 16848481
Free inodes: 1014969
First block: 0
Block size: 2048
Fragment size: 2048
Reserved GDT blocks: 512
Blocks per group: 16384
Fragments per group: 16384
Inodes per group: 128
Inode blocks per group: 8
Filesystem created: Tue Jul
Last mount time: Thu Aug 21 05:58:25 2008
Last write time: Fri Jan
Mount count: 2
Maximum mount count: 20
Last checked: Tue Jul
```

```
Check interval: 15552000 (6 months)
Next check after: Sat Dec 27 23:06:03 2008
Reserved blocks uid: 0 (user root)
Reserved blocks gid: 0 (group root)
First inode: 11
Inode size: 128
Default directory hash: tea
Directory Hash Seed: 12345829-1236-4123-9aaa-ccccc123292b
```

完全复制粘贴的作者的磁盘信息. 感觉用处不是很大的样子... 知道有这么个东西就好.

新建一个swap分区

首先, 你知道什么是交换分区(swap)嘛?

不知道的话先去查查吧!

如果内存不是很紧张, 也不需要休眠的话, `Swap` 分区在个人主机上存在的意义已经不是很大了.我自己是一块120G的SSD, 所以硬盘很吃紧, 分出8G的大小给 `Swap` 太不划算了, 而且内存已经足够用了, 所以我就取消了 `Swap` 分区, 这样看起来会安心一些 :P

但是在 `VPS` 主机上, 交换分区还是很重要的, 因为私人买的主机, 一般内存都不会很大(因为贵啊!), 建立交换分区就会把系统暂时不用的数据存起来, 系统也就会有更多的内存来处理当前事务. 有的主机提供商会自动划分交换分区, 有的则不然(比如DigitalOcean), 需要你自己建立.

首先新建一个够你用的文件:

```
# dd if=/dev/zero of=/home/swap-fs bs=1M count=512
512+0 records in
512+0 records out
# ls -l /home/swap-fs
-rw-r--r-- 1 root root 536870912 Jan 2 23:13 /home/swap-fs
```

要不要科普一下 `/dev/zero` 呢? 你还是自己去查吧 :) -> 我的谷歌

哦对了,还有我的维基

然后把新建的文件转换为 `swap` 格式:

```
# mkswap /home/swap-fs
```

挂载我们新建的 `swap` 分区:

```
swapon /home/swap-fs
```

如果想长期使用, 那就写入到 `/etc/fstab` :

```
/home/swap-fs swap swap defaults 0 0
```

新建用户

建立用户时只指定用户名:

```
# useradd jsmith
```

然后别的都按照默认设置, 需要注意的是, 这样建立的用户没有默认密码的, 需要新给他设置密码.

建立用户时附带信息:

```
# adduser -c "John Smith - Oracle Developer" -e 12/31/09 jsmith
```

其中:

- `-c` 描述用户
- `-e` 用户过期时间(这样说有点扯, 大概就是到了这个时间, 这个用户就不能用了)

改用户的密码用: `passwd user_name` 这么基础的就不必啰嗦了.

最后是查看默认添加的用户信息:

```
# useradd -D
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/bash
SKEL=/etc/skel
```

这里作者没有说太多, 不是江郎才尽, 而是这里真没什么好说的, 需要注意的一点是, `useradd` 和 `adduser` 是不一样的. 具体哪里不一样? 你试试就知道了!

还有哦, `deluser` 和 `userdel` 也不一样哦~ 这里倒是挺有趣的.

新建用户组

新建一个用户组:

```
# groupadd developers
```

把用户添加到用户组:

```
usermod -g developers jsmith
```

(不忍再吐槽一句, 这也叫技巧? 明明是基本操作啊! 以后翻译东西一定要取其精华, 去其糟粕.)

大家整理东西的时候也是, 并不是别人写的都是好的, 有时候要选择性的抛弃一些东西, 选择性的吸收一些东西.

SSH密钥登录

密码登录不安全, 万一被爆破了呢? 要假设黑客无所不能(除了破解2048位的密钥,哈哈哈)!

如果你没有密码学的相关知识也没关系, 我可以大体举个例子说明以下. 所谓公钥, 相当于一把锁, 那么私钥呢, 就是打开这把锁的钥匙了, 而且, 这把锁的钥匙只有一个, 这把钥匙也只能打开这一把锁. 我们可以理解为,钥匙的复杂度很高很高, 高到无法破解. 解密的时候呢, 公钥那边给出一个问题, 这个问题只有私钥能够解开, 拥有私钥的你在本地把题目解开之后, 把结果发送给服务器, 服务器那边一看, 哟, 答对了, 放行! 然后你就成功登陆了.

(过程大概就是这个意思, 其中涉及到密码学公钥系统的知识, 我学的不好, 只能理解个大概, 所以各位看官也就听个大概吧, 小弟哪里讲的不对, 欢迎指正!)

第一步,创建私钥和公钥:

```
root@key:~# ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Created directory '/root/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
91:27:cf:01:0c:64:90:dc:08:d0:f6:fa:be:a8:88:91 root@key
The key's randomart image is:
+--[ RSA 2048]----+
|.o.o.*+o.        |
|  o +.. .o       |
| . .    + o      |
|    .    * .     |
|   .    S o      |
| ..              |
|E  .             |
|o. ..            |
|+...o.           |
+-----------------+
root@key:~#
```

就一个命令(`ssh-keygen`).

然后你就得到了你的公钥和私钥:

```
root@key:~# ls .ssh/
id_rsa  id_rsa.pub
root@key:~#
```

其中公钥(`.pub` 结尾的文件)可以随意分发, 但是私钥一定要保存好, 不能通过不安全的方式带到别处.(什么是不安全的方式? 看你的安全等级咯~)

第二步, 把公钥拷贝到远程主机

```
jsmith@local-host$ ssh-copy-id -i ~/.ssh/id_rsa.pub remote-host
jsmith@remote-host’s password:
Now try logging into the machine, with "ssh 'remote-host'",
and check in: .ssh/authorized_keys to make sure we haven't
added extra keys that you weren't expecting.
```

这样就把公钥拷贝上去啦，然后确认自己是否可以免密码登陆服务器，如果可以，就进入第三步，如果不行呢，那就再复制一遍.

第三步，关闭远程服务器的密码登陆

这一步可选，但是我还是建议你这样做.

```
root@key:~# cat /etc/ssh/sshd_config | grep "PasswordAuthenticat
ion yes"
#PasswordAuthentication yes
root@key:~#
```

把这里改成 `no` ,并取消注释，重启ssh服务.

嗯... 如果你不想用 `ssh-copy-id` 的话，直接把公钥复制到远程主机的 `~/.ssh/authorized_keys` 这个文件里也是可以的，但是要保证一字不差哟~而且这种方法很方便，如果你有多台主机，就可以写一个脚本自己写入公钥，改写端口，以及禁止密码登录.

ssh-copy-id 和 ssh- agent

看了半天也不知道这是什么鬼技巧，大概就是说了一个问题，然后怎样解决这个问题.

翻译模式开启：

如果没有 `-i` 的参数或者 `~/.ssh/identity.pub` 不可用，`ssh-copy-id` 就会报错：

```
jsmith@local-host$ ssh-copy-id -i remote-host
/usr/bin/ssh-copy-id: ERROR: No identities found
```

如果你用 `ssh-add` 加载了公钥到 `ssh-agent` 的话，`ssh-copy-id` 就会 `ssh-agent` 那里得到公钥并且拷贝到远程服务器.

```
jsmith@local-host$ ssh-agent $SHELL
jsmith@local-host$ ssh-add -L
The agent has no identities.
jsmith@local-host$ ssh-add
Identity added: /home/jsmith/.ssh/id_rsa
(/home/jsmith/.ssh/id_rsa)
jsmith@local-host$ ssh-add -L
ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEAsJIEILxftj8aSxMa3d8t6JvM79D
aHrtPhTYpq7kIEMUNzApnyxsHpH1tQ/Ow==
/home/jsmith/.ssh/id_rsa
jsmith@local-host$ ssh-copy-id -i remote-host
jsmith@remote-host's password:
Now try logging into the machine,
... ... ...
```

首先我们看到，`ssh-add -L` 没有回显任何已有的公钥，添加以后，再用 `ssh-copy-id` 就可以以默认的公钥拷贝到远程主机上了.

也许作者在这里是想说明这几个软件之间的关系，尽管有别的方法拷贝，或者补全参数就能搞定.

(但我相信作者是在这里充数啊!!! -.-)

Crontab 命令

这一小节是介绍计划任务的, 对, 没错, 就是定时爆炸的.

怎样像 `cron` 添加一个计划任务

比较简单的写法:

```
➤ crontab -e
0 5 * * * /root/bin/backup.sh
```

敲入 `crontab -e` 之后会打开一个新文件, 输入什么呢? 输入你想执行的计划任务.

解释一下这些东西:

```
{minute} {hour} {day-of-month} {month} {day-of-week} {full-path-
to-shell-script}
```

- `minute` 0-59
- `hour` 0-23
- `day-of-month` 0-31
- `month` 1-12
- `day-of-week` 0-7

这些英文这么简单, 不用翻译了吧 :)

举栗子

1.每天 `00:01` 运行备份脚本

```
1 0 * * * /root/bin/backup.sh
```

2.周一到周五每天 `11:59` 运行备份脚本

```
59 11 * * 1,2,3,4,5 /root/bin/backup.sh
```

等价于:

```
59 11 * * 1-5 /root/bin/backup.sh
```

3.每5分钟运行某个脚本

```
*/5 * * * * /root/bin/check-status.sh
```

4.每月1号 `13:10` 运行某个脚本

```
10 13 1 * * /root/bin/full-backup.sh
```

crontab 命令参数

- `crontab -e` 编辑任务列表
- `crontab -l` 列出所有任务
- `crontab -r` 移除任务文件(删除所有任务)
- `crontab -ir` 删除文件的时候让你确认(跟 rm -i 一样)

扩展阅读

- 15 Awesome Cron Job Examples
- How to Run Cron Every 5 Minutes, Seconds, Hours, Days, Months
- Cron Vs Anacron

用SysRq key安全的重启

Magic SysRq key 是 Linux 内核中的一个组合键, 它允许用户执行一些低权限的命令, 无视系统当前的状态.

它经常被用来恢复死机的系统, 或者重启系统而不打断文件系统的状态. 这个组合键是 `Alt+SysRq+commandkey` ,在很多系统中 `SysRq` 键是 `printscreen` 键.

首先, 你要启用这个功能:

```
echo "1" > /proc/sys/kernel/sysrq
```

`commandkey` 列表

- `k` 杀死所有绑定在当前虚拟终端上的进程.
- `s` 同步所有已挂载的文件系统.
- `b` 立即重启系统, 不同步数据也不卸载磁盘.
- `e` 发送 SIGTERM 信号到所有进程(除了init主进程).
- `m` 向终端输出当前的内存信息.
- `i` 发送 SIGKILL 信号到所有进程(除了init主进程).
- `r` 将键盘从 raw mode 转换到 XLATE mode.
- `t` 向终端输出当前所有的任务(进程)信息.
- `u` 以只读模式重新挂载当前已经挂载的文件系统.
- `o` 立即关闭系统.
- `p` 打印当前的注册信息和标志位(不知道是啥..).
- `0-9` 设定终端日志级别, 控制发送到终端的内核信息.
- `f` 杀死进程的(消耗更多内存).
- `h` 显示帮助信息.

另注: **Ubuntu**上并没有成功... 所以如果你觉着上面的是在扯淡或者没有卵用, 那也是很正常的, 因为我也这样觉着...

Parted 命令

知道 `Gparted` 吧? 这个 `parted` 就是它的无图形化界面.

我还是推荐大家用 `Gparted`, 因为操作很直观嘛. 当然, 学了 `parted` 更好啦, 有些时候没有图形化界面, 就得需要命令行界面了.

还要记住一点, 每次操作的时候, 你必须清楚地明白你在做什么, 否则没有后悔药给你吃的.

你也知道数据恢复是多么的昂贵.

正餐开始(看看就好, 不必上手操作的):

选择要操作的分区

当你直接输入 `parted` 的时候, 默认的操作对象是系统中第一个可用的分区.

下面的例子中, 它自动选择了 `/dev/sda` 做为操作对象, 因为这是系统中第一个可用的分区.

```
# parted
GNU Parted 2.3
Using /dev/sda
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted)
To choose a different hard disk, use the select command as shown
 below.
(parted) select /dev/sdb
```

当他找不到所选择的分区时, 就会抛出一个错误:

```
Error: Error opening /dev/sdb: No medium found
Retry/Cancel? y
```

用 `print` 显示所有可操作的磁盘

用这个命令可以显示出所有可操作的磁盘分区, 他还能够显示一些磁盘的属性, 比如模式, 大小, 蔟大小, 分区表信息等等.

```
(parted) print
Model: ATA WDC WD5000BPVT-7 (scsi)
Disk /dev/sda: 500GB
Sector size (logical/physical): 512B/4096B
Partition Table: msdos
Number Start End Size Type Filesystem Flags
1 1049kB 106MB 105MB primary fat16 diag
2 106MB 15.8GB 15.7GB primary ntfs boot
3 15.8GB 266GB 251GB primary ntfs
4 266GB 500GB 234GB extended
5 266GB 269GB 2682MB logical ext4
7 269GB 270GB 524MB logical ext4
8 270GB 366GB 96.8GB logical lvm
6 366GB 370GB 3999MB logical linux-swap(v1)
9 370GB 500GB 130GB logical ext4
```

用 `mkpart` 在硬盘中创建主分区

`mkpart` 命令可以创建主分区和逻辑分区, 他所需要的参数是 `START` 和 `END` . 也就是标记从哪儿分, 分到哪儿. 单位是 `MB` .

下面的例子分了一个15G的区:

```
(parted) mkpart primary 106 16179
```

你也可以用下面的例子来开启磁盘的 `boot` 标志, 使之作为一个启动硬盘. Linux保留了1-4或者1-3作为主分区, 把其他的数字所谓扩展分区.

```
(parted) set 1 boot on
```

用 `mkpart` 在硬盘中创建逻辑分区

在执行操作之前, 我们先看看磁盘的信息:

```
(parted) print
Model: ATA WDC WD5000BPVT-7 (scsi)
Disk /dev/sda: 500GB
Sector size (logical/physical): 512B/4096B
Partition Table: msdos
Number Start End Size Type Filesystem Flags
1 1049kB 106MB 105MB primary fat16 diag
2 106MB 15.8GB 15.7GB primary ntfs boot
3 15.8GB 266GB 251GB primary ntfs
4 266GB 500GB 234GB extended
5 266GB 316GB 50.0GB logical ext4
6 316GB 324GB 7999MB logical linux-swap(v1)
7 324GB 344GB 20.0GB logical ext4
8 344GB 364GB 20.0GB logical ext2
```

用下面的命令来创建一个128G的逻辑分区:

```
(parted) mkpart logical 372737 500000
```

现在我们再来查看一下磁盘信息:

```
(parted) print
Model: ATA WDC WD5000BPVT-7 (scsi)
Disk /dev/sda: 500GB
Sector size (logical/physical): 512B/4096B
Partition Table: msdos
Number Start End Size Type Filesystem Flags
1 1049kB 106MB 105MB primary fat16 diag
2 106MB 15.8GB 15.7GB primary ntfs boot
3 15.8GB 266GB 251GB primary ntfs
4 266GB 500GB 234GB extended
5 266GB 316GB 50.0GB logical ext4
6 316GB 324GB 7999MB logical linux-swap(v1)
7 324GB 344GB 20.0GB logical ext4
8 344GB 364GB 20.0GB logical ext2
9 373GB 500GB 127GB logical
```

用 `mkfs` 创建文件系统

有了磁盘还不算完，还要在里面创建文件系统，当然，这种操作也是极其危险的，因为格式化之后的数据将会全部丢失。`Parted` 支持的文件系统有:ext2, mips, fat16, fat32, linux-swap 和 reiserfs(如果安装了libreiserfs的话).

下面的例子把第八块儿分区的格式从 `ext4` 变为了 `ext2` :

```
(parted) print
Model: ATA WDC WD5000BPVT-7 (scsi)
Disk /dev/sda: 500GB
Sector size (logical/physical): 512B/4096B
Partition Table: msdos
Number Start End Size Type Filesystem Flags
1 1049kB 106MB 105MB primary fat16 diag
2 106MB 15.8GB 15.7GB primary ntfs boot
3 15.8GB 266GB 251GB primary ntfs
4 266GB 500GB 234GB extended 5 266GB 316GB 50.0GB logical ext4
6 316GB 324GB 7999MB logical linux-swap(v1)
7 324GB 344GB 20.0GB logical ext4
8 344GB 364GB 20.0GB logical ext4
9 364GB 500GB 136GB  logical ext4
```

然后开始格式化:

```
(parted) mkfs
Warning: The existing file system will be destroyed and
all data on the partition will be lost. Do you want to
continue?
Yes/No? y
Partition number? 8
File system type?
[ext2]? ext2
```

执行完成之后我们再来看一下:

```
(parted) print
Model: ATA WDC WD5000BPVT-7 (scsi)
Disk /dev/sda: 500GB
Sector size (logical/physical): 512B/4096B
Partition Table: msdos
Number Start End Size Type Filesystem Flags
1 1049kB 106MB 105MB primary fat16 diag
2 106MB 15.8GB 15.7GB primary ntfs boot
3 15.8GB 266GB 251GB primary ntfs
4 266GB 500GB 234GB extended 5 266GB 316GB 50.0GB logical ext4
6 316GB 324GB 7999MB logical linux-swap(v1)
7 324GB 344GB 20.0GB logical ext4
8 344GB 364GB 20.0GB logical ext2
9 364GB 500GB 136GB  logical ext4
```

(没错我就是复制上面的内容然后再把4改成2的.)

用 `mkpartfs` 既划分磁盘又对分区进行格式化

其实就是把上面的两条命令组合了起来, 可以这么理解.

先是看一下原分区什么样:

```
(parted) print
Model: ATA WDC WD5000BPVT-7 (scsi)
Disk /dev/sda: 500GB
Sector size (logical/physical): 512B/4096B
Partition Table: msdos
Number Start End Size Type Filesystem Flags
1 1049kB 106MB 105MB primary fat16 diag
2 106MB 15.8GB 15.7GB primary ntfs boot
3 15.8GB 266GB 251GB primary ntfs
4 266GB 500GB 234GB extended 5 266GB 316GB 50.0GB logical ext4
6 316GB 324GB 7999MB logical linux-swap(v1)
7 324GB 344GB 20.0GB logical ext4
8 344GB 364GB 20.0GB logical
```

然后我们新建一个分区并格式化:

```
(parted) mkpartfs logical fat32 372737 500000
```

然后再看看新的磁盘信息:

```
(parted) print
Model: ATA WDC WD5000BPVT-7 (scsi)
Disk /dev/sda: 500GB
Sector size (logical/physical): 512B/4096B
Partition Table: msdos
Number Start End Size Type Filesystem Flags
1 1049kB 106MB 105MB primary fat16 diag
2 106MB 15.8GB 15.7GB primary ntfs boot
3 15.8GB 266GB 251GB primary ntfs
4 266GB 500GB 234GB extended 5 266GB 316GB 50.0GB logical ext4
6 316GB 324GB 7999MB logical linux-swap(v1)
7 324GB 344GB 20.0GB logical ext4
8 344GB 364GB 20.0GB logical 9 373GB 500GB 127GB logical fat32 l
ba
(parted)
```

用 `resize` 来改变分区大小

```
(parted) resize 9
Start? [373GB]? 373GB
End? [500GB]? 450GB
```

很简单的, 选中要操作的对象, 然后选择开始, 在选择结束, 就OK了.(其实用 `Gparted` 更简单, 直接拖动就好了...)

然后下面还是一堆磁盘信息, 节省篇幅, 我就不贴了.

用 `cp` 来复制分区

这个命令有点厉害, 类似于 `dd` , 他可以把整个分区都复制到别处(别的分区). 需要注意的是, 保存被复制的分区的大小一定要足够, 还有, 这是一个覆盖操作.

(作者又打印了一遍磁盘信息, 我在此略过)

建议把要操作的两块磁盘都先卸载，以免有其他的程序对他们进行读写.

```
# mount /dev/sda8 /mnt
# cd /mnt
# ls -l
total 52
-rw-r--r-- 1 root root
-rw-r--r-- 1 root root
0 2011-09-26 22:52 part8
20 2011-09-26 22:52 test.txt

# umount /mnt
# mount /dev/sda10 /mnt
# cd /mnt
# ls -l
total 48
-rw-r--r-- 1 root root 0 2011-09-26 22:52 part10
```

用 `cp` 命令把8复制到10:

```
(parted) cp 8 10
growing file system... 95% (time left 00:38)
```

下面是复制完成之后的结果:

```
# mount /dev/sda10 /mnt
# cd /mnt
# ls -l
total 52
-rw-r--r-- 1 root root 0 2011-09-26 22:52 part8
-rw-r--r-- 1 root root 20 2011-09-26 22:52 test.txt
```

另外需要注意的是，如果目标磁盘和被复制的磁盘的文件系统不一致，这个复制操作将会重建目标磁盘的文件系统使其和被复制的磁盘分区一样.(也就是说，复制操作使得两个分区一模(mu)一样.)

用 `rm` 命令移除某个分区

```
(parted) rm
Partition number? 9
(parted)
```

也很简单...

这样一删的话，里面的东西就全没了 :)

Rsync 命令

`rsync` 表示 `remote sync` .

也就是远程同步的意思.

它是用来备份的.

`rsync` 的特性

- 速度快: 第一次同步的时候会将全部的文件都进行备份, 以后的时候他就回对比改变过的文件, 而使得备份速度很快.
- 安全: `rsync` 允许在同步的时候使用ssh协议进行加密数据.
- 消耗带宽小: 同步过程中使用压缩技术, 使得传输的数据更小.
- 权限低: 使用 `rsync` 不需要任何额外的权限.

语法:

```
$ rsync options source destination
```

源地址和目的地址既可以是本地也可以时远程服务器, 如果是远程服务器的话还需要指定登录名以及远程服务器地址.

在本地同步两个文件夹

```
$ rsync -zvr /var/opt/installation/inventory/ /root/temp
building file list ... done
sva.xml
svB.xml
.
sent 26385 bytes
received 1098 bytes
total size is 44867
speedup is 1.63
```

再上面例子的参数中:

- `-z` 开启压缩
- `-v` 输出日志信息
- `-r` 递归同步(在文件夹中)

还有一点需要说明, `rsync` 没有保留原始文件的创建时间信息, 也就是说, 目的地的文件的创建时间与原始文件的创建时间不一致.

保存文件的创建时间

刚说了他不能保留原始文件的创建时间, 这就过来打脸了:

`-a` 选项可以使得原始文件与备份文件一模一样, 包括创建时间, 属性, 所属用户和所属组, 权限信息等等.

这里我就不演示了, 你可以自己试一下 :D

同步文件到远程服务器

```
rsync -avz /root/temp/ thegeekstuff@192.168.200.10:/home/thegeek
stuff/temp/
Password:
building file list ... done
./
rpm/
rpm/Basenames
rpm/Conflictname
sent 15810261 bytes received 412 bytes 2432411.23 bytes/sec
total size is 45305958
speedup is 2.87
```

同步远程目录的格式为:

```
rsync -avz [local_path] [username]@[server_ip_address]:[/file_pa
th]
```

同步远程服务器文件到本地

这个跟上一个是反方向操作.不过命令格式都差不多的:

```
rsync -avz [username]@[server_ip_address]:[/file_path] [local_pa
th]
```

没什么特别复杂的地方，哦对了，同步都是覆盖操作的，没有像git那样有记录什么的，所以还是小心一点咯

扩展阅读

- 15 rsync Command Examples
- 6 rsync Examples
- 更多介绍 - 中文

Chkconfig 命令 / Service 命令

Ubuntu上并没有这条命令...

所以我就说一下**Ubuntu**上取而代之的 `service` 命令吧 :)

首先说一下他是干嘛的, 字如其名, 当然是用来控制服务的.

查看当前所有服务状态

在 `man service` 中我们可以看到有 `service --status-all` 这么一条命令, 没错, 他就是用来查看当前系统所运行的所有服务状态的.

输出的结果如下:

```
➤ service --status-all
 [ + ]  acct
 [ + ]  acpid
 [ + ]  alsa-utils
 [ - ]  anacron
 [ + ]  apparmor
 [ + ]  apport
 [ + ]  atd
 [ + ]  avahi-daemon
 [ + ]  binfmt-support
 [ - ]  bluetooth
 [ - ]  bootmisc.sh
 [ - ]  brltty
 [ + ]  cgmanager
 [ - ]  cgproxy
 [ - ]  cgroupfs-mount
 [ - ]  checkfs.sh
 [ - ]  checkroot-bootclean.sh
 [ - ]  checkroot.sh
... ... ...
 [ + ]  urandom
 [ - ]  uuidd
 [ + ]  virtualbox
 [ + ]  whoopsie
 [ - ]  x11-common
```

列举的就是所有的服务，其中,每个服务名前面的 `+` 代表服务正在运行，`-` 则说明服务没有在运行.

系统服务开启与关闭

语法:

```
Usage: service < option > | --status-all | [ service_name [ comm
and | --full-restart ] ]
```

具体还要看服务的脚本是怎么写的，不过一般都会有 `start` , `stop` , `status` 这三个.

脚本的存放位置在 `/etc/init.d/` 和 `/etc/systemd/system/` 目录.

添加服务到开机启动

这里用到的则是另一个命令了: `update-rc.d`

设置某项服务开机启动:

```
update-rc.d [service_name] defaults
```

删除某项服务开机启动:

```
update-rc.d -f [service_name] remove
```

当然, 这些命令都需要root权限.

Anacron 配置

`anacron` 是 `cron` 在桌面系统中的软件.(其实桌面系统也有cron.)

你可能会问, 为什么要有一个桌面系统版本啊? 因为, 桌面系统不是服务器, 不需要也不可能24小时运行, 所以 `anacron` 的意义就是在一个非24小时运行的机器上执行计划任务.

比如说, 当你在笔记本上运行了一个每天晚上11点备份文件的脚本, 但是你不可能确保每天晚上11点都会开机啊, 那么备份就不会运行了么? 对于 `cron` 来说是的, 因为过了那个点儿了, 他就不会运行了, 但是对于 `anacron` 来说就不是, 因为他就是专门应付这种情况的, 如果今天晚上11点没有运行备份脚本, 那么当你再次启动计算机的时候, 他就会立即执行这个备份脚本. 同样, 如果系统待机也会是这样, 当系统再次被唤起的时候就会执行那些本该执行却没有执行的计划任务.

Anacrontab格式

正如 `cron` 的记录文件是 `/etc/crontab` 一样, `anacron` 也有一个记录文件 `/etc/anacrontab` .

`/etc/anacrontab` 的格式为:

```
period delay job-identifier command
```

第一个字段是执行的周期, 如果写1,就表示每天执行, 如果写7就代表每周执行, 同理, 如果写30,就代表每月执行, 当然, 也可以是任意的阿拉伯数字.

第二个字段是延迟执行的时间, 这个是用来应付那些没有被正常执行的计划任务的, 也就是说, 如果一个任务没有被正常执行(关机待机等情况), 当再次开机后, 需要延迟多长时间再次执行. 它的单位是分钟.

第三个字段是任务的标识**(zhi4)**符, 每个文件都必须又一个独一无二的标识符, 标识符文件保存在 `/var/spool/anacron/` 目录下, 每个文件记录着不同任务上次执行的时间, 这也就解释的通为什么它可以在开机后执行那些没有被正常执行的任务了.

第四个字段是要执行的命令, 比如要执行一个脚本, 就可以写" `/bin/sh /root/bala/backup.sh` ".

举个栗子

```
➤ cat /etc/anacrontab
7 15 test.daily /bin/sh /root/bala/backup.sh
```

每七天执行备份脚本，任务的标识符为 `test.daily`，如果没有正常运行，则在开机后15分钟之后再次运行.

两个变量

`START_HOURS_RANGE` 和 `RANDOM_DELAY`

干什么用呢? 慢慢说.

上面的这个例子是，每天执行一个脚本，什么时候执行? 开机十五分钟之后执行. 但是，如果你的笔记本一周不关机怎么办? 难道备份脚本就不要执行了吗? 肯定不行啊. 所以这个时候就需要 `/etc/anacrontab` 里的变量(`START_HOURS_RANGE`)了.

默认的值是3-22，也就是凌晨3点到晚上10点，这个时间段所谓一个开机周期，如果过了这个周期，那 `anacron` 就会知道新的一天开始了.

上面提到的第二个变量也有点意思，刚才我们说到开机15分钟之后执行那个脚本，但是真实情况还要加点东西，加点什么呢? 加的就是这个 `RANDOM_DELAY` 内的分钟数，默认的 `RANDOM_DELAY` 值是45，也就是说，每个任务开启的时间是不确定的，是在用户指定的时间上在加0-45之间的一个数.

Cron Vs Anacron

Cron	Anacron
最小执行周期是分钟	最小执行周期是天
任何用户都可以运行	只允许超级用户运行
要求系统7x24小时运行，如果某个任务错过了执行时间，那么它将不会被执行	不要求系统7x24小时运行. 如果某个计划任务没有被正常运行，那么他将在下次开机后再次运行
服务器	桌面机或者笔记本

扩展阅读: Cron Vs Anacron

IPTables 规则举例

删除现有的规则

用下面的命令来清空当前所设置的所有规则.

```
iptables -F
(or)
iptables --flush
```

设置默认的链规则

默认的规则是允许通过, 如果你有特殊需求, 可以设置成拒绝:

```
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT DROP
```

当你设置了默认的INPUT 和 OUTPUT 链为DROP时, 在你的所有防火墙上必须要再定义两条规则, 一个是入, 另一个是出.

下面的所有例子中, 每一个方案都定义了两条规则, 一条是入规则, 另一条则是出规则.(因为默认是拒绝通过出入包的.)

当然, 如果你足够相信你的用户, 也可以将默认的出站规则设置为接受, 允许所有的出站请求.这样的话, 防火墙就只需要一条入站规则了.

拒绝某个特定的ip地址

在列举更多的例子之前, 先来看一下怎样拒绝来自某个ip的全部请求:

```
iptables -A INPUT -s "ip_address_to_block" -j DROP
```

当你在日志中发现某个ip异常时, 你就可以用这条规则来限制这个ip地址.

你也可以用下面的方式来拒绝eth0网卡上来自这个ip地址的tcp请求:

```
iptables -A INPUT -i eth0 -s "ip_address_to_block" -j DROP
iptables -A INPUT -i eth0 -p tcp -s "ip_address_to_block" -j DRO
P
```

允许所有的SSH入站请求

```
iptables -A INPUT -i eth0 -p tcp --dport 22 -m state --state NEW
,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --sport 22 -m state --state ES
TABLISHED -j ACCEPT
```

上面的规则允许了所有eth0网卡上的SSH入站请求.

限制特定的ip建立SSH连接

```
iptables -A INPUT -i eth0 -p tcp -s 192.168.100.0/24 --dport 22
-m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --sport 22 -m state --state ES
TABLISHED -j ACCEPT
```

上面的例子仅允许来自192.168.100.*的连接.

当然也可以把 `192.168.100.0/24` 换成 `192.168.100.0/255.255.255.0` , 如果你不嫌麻烦的话...

扩展阅读

- 25 Most Frequently Used Linux IPTables Rules Examples
- IPTables Tables, Chains, Rules Fundamentals
- How to Add Firewall Rules
- Incoming and Outgoing Rule Examples

都是英文的, 一点一点啃吧 :)

第九章 - 安装软件

这一章介绍了在不同的发行版上用不同的方法安装软件.

RedHat系列(rpm),Debian系列(apt-*).还有CentOS的yum.

具体的区别可以看 What is the difference between yum, apt-get, rpm, ./configure && make install?

发行版	低级管理	高级管理
Debian and derivatives	dpkg	apt-get / aptitude
CentOS	rpm	yum
openSUSE	rpm	zypper

最后一节介绍了怎样通过源码进行安装.

这里介绍的可能没有以前那么好玩了,不过都很实用(对每个发行版来说).

~~还有80页就翻译完啦~~~

Yum 命令

Yum 的全称是 "Yellowdog Updater Modified".

wikipedia链接

用 `yum install` 安装软件包

如果想安装某个软件, 只需要这样做:

```
yum install packagename
```

下面的例子安装了postgresql:

```
# yum install postgresql.x86_64
Resolving Dependencies
Install 2 Package(s)
Is this ok [y/N]: y
Running Transaction
Installing : postgresql-libs-9.0.4-5.fc15.x86_64 1/2
Installing : postgresql-9.0.4-5.fc15.x86_64 2/2
```

默认情况下, `yum` 会向你确认要安装的包, 如不想显示这个确认, 可以加一个 `-y` 的参数:

```
# yum -y install postgresql.x86_64
```

用 `yum remove` 卸载软件包

卸载软件包也很简单:

```
# yum remove
postgresql.x86_64
Package postgresql.x86_64 0:9.0.4-5.fc15 will be erased
Is this ok [y/N]: y
Running Transaction
Erasing : postgresql-9.0.4-5.fc15.x86_64 1/1
```

这个例子是卸载postgresql的.

用 `yum update` 升级某个软件包

```
# yum update postgresql.x86_64
```

这个例子升级了postgresql.

用 `yum search` 搜索某个软件包

如果你不确定具体要安装的软件包的名字,可以用 `yum search` 来搜索.

下面的例子搜索了那些名字中带有firefox的包:

```
# yum search firefox
Loaded plugins: langpacks, presto, refresh-packagekit
============== N/S Matched: firefox ======================
firefox.x86_64 : Mozilla Firefox Web browser
gnome-do-plugins-firefox.x86_64
mozilla-firetray-firefox.x86_64
mozilla-adblockplus.noarch : Mozilla Firefox extension
mozilla-noscript.noarch : Mozilla Firefox extension
```

如果想显示全部的信息,可以用 `yum search all` .

用 `yum info` 来显示包的所有信息

当你通过 `yum search` 搜索到某个包时,可以用 `yum info` 来查看包的信息:

```
# yum info samba-common.i686
Loaded plugins: langpacks, presto, refresh-packagekit Available
Packages
Name        : samba-common
Arch        : i686
Epoch       : 1
Version     : 3.5.11
Release     : 71.fc15.1
Size        : 9.9 M
Repo        : updates
Summary     : Files used by both Samba servers and clients
URL         : http://www.samba.org/
License     : GPLv3+ and LGPLv3+
Description : Samba-common provides files necessary for both the
 server and client
```

上面的例子查看了 `samba-common` 的包信息.

扩展阅读

[15 Linux Yum Command Examples]

RPM 命令

`RPM` 的全称是 `Red Hat Package Manager` .

用 `rpm -ivh` 来安装RPM包

RPM的文件名包含了软件的名字, 版本号, 发行号,以及软件架构.

比如在 `MySQL-client-3.23.57-1.i386.rpm` 这个文件名中,

- MySQL-client 是包的名字
- 3.23.57 是版本号
- 1 是发行号
- i386 是软件架构(32位)

当你安装一个RPM包时, RPM会检查你的系统是否能够安装这个包, 看一下这个包文件要安装在哪儿, 安装完成之后还会更新RPM的软件数据库.

```
# rpm -ivh
MySQL-client-3.23.57-1.i386.rpm
Preparing...########################################### [100%]
1:MySQL-client##################################### [100%]
```

上面的例子中:

- `-i` 代表安装(install)
- `-v` 代表显示详细信息(verbose)
- `-h` 打印hash marks(我也不知道这是什么鬼...)

用 `rpm -qa` 列出全部已经安装的包

```
# rpm -qa
cdrecord-2.01-10.7.el5
bluez-libs-3.7-1.1
setarch-2.0-1.1
...
...
...
```

其中:

- `-q` 列举, 查询(query)
- `-a` 全部(all)

列举包的时候规定显示格式

```
# rpm -qa --queryformat '%{name-%{version}-%{release} %{size}\n'
cdrecord-2.01-10.7 12324
bluez-libs-3.7-1.1 5634
setarch-2.0-1.1 235563
...
...
...
```

用 `rpm -qf` 查看某个文件所属的包

```
# rpm -qf /usr/bin/mysqlaccess
MySQL-client-3.23.57-1
```

可以看到这个文件属于 `MySQL-client-3.23.57-1` 这个包.

用 `rpm -qip` 查看某个已安装包的具体信息

```
# rpm -qip MySQL-client-3.23.57-1.i386.rpm
Name        : MySQL-client Relocations: (not relocatable)
Version     : 3.23.57
Vendor      : MySQL AB
Release     : 1
Build Date  : Mon 09 Jun 2003
Install Date:
Build Host  : build.mysql.com
Group       : Applications/Databases
Size        : 5305109
Signature   : (none)
Packager    : Lenz Grimmer
URL         : http://www.mysql.com/
Summary     : MySQL - Client
License     : GPL / LGPL
Description : This package is a standard MySQL client.
```

其中:

- `-i` 查看rpm包的信息
- `-p` 指定包的名字

查看包内的文件

```
$ rpm -qlp ovpc-2.1.10.rpm
/usr/bin/mysqlaccess
/usr/bin/mysqldata
/usr/bin/mysqlperm
...
...
/usr/bin/mysqladmin
```

其中 `-l` 的意思就是列出包内的文件.

用 `rpm -qRP` 查看某个包依赖的其他包

```
# rpm -qRp MySQL-client-3.23.57-1.i386.rpm
/bin/sh
/usr/bin/perl
```

这个 `mysql-client` 就依赖 `sh` 和 `perl` .

扩展阅读

[RPM Command: 15 Examples to Install, Uninstall, Upgrade, Query RPM Packages]

Apt-* 命令

啊哈，终于到了Debian系列啦~

`apt-cache search` 搜索软件包

```
➤ apt-cache search ^apache2$
apache2 - Apache HTTP Server
```

这个例子搜索了apache2这个软件，可以看到，支持正则表达式哦~

`dpkg -l` 列出当前安装的软件包

其实也不仅列出了安装的软件包，还列出了曾经安装过，现在没有完全卸载的软件包，前面有几个标识符，比如 `i` ，`r` ：

```
apache2 - Apache HTTP Server
➤ dpkg -l
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-a
Wait/Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name                                    Version
 Architecture            Description
+++-====================================-=======================
-=======================-=======================================
======================================
ii  account-plugin-aim                  3.12.10-0ubuntu2
 amd64                   Messaging account plugin for AIM
ii  account-plugin-facebook             0.12+15.10.20150723-0ub
 all                     GNOME Control Center account plugin for
 single signon - facebook
ii  account-plugin-flickr               0.12+15.10.20150723-0ub
 all                     GNOME Control Center account plugin for
 single signon - flickr
ii  account-plugin-google               0.12+15.10.20150723-0ub
 all                     GNOME Control Center account plugin for
```

```
 single signon
ii  account-plugin-jabber                       3.12.10-0ubuntu2
 amd64                  Messaging account plugin for Jabber/XMP
P
ii  account-plugin-salut                        3.12.10-0ubuntu2
 amd64                  Messaging account plugin for Local XMPP
 (Salut)
ii  account-plugin-yahoo                        3.12.10-0ubuntu2
 amd64                  Messaging account plugin for Yahoo!
ii  accountsservice                             0.6.40-2ubuntu5
 amd64                  query and manipulate user account infor
mation
ii  acct                                        6.5.5-2.1ubuntu1
 amd64                  The GNU Accounting utilities for proces
s and login accounting
ii  ack-grep                                    2.14-4
 all                    grep-like program specifically for larg
e source trees
ii  acl                                         2.2.52-2
 amd64                  Access control list utilities
ii  acpi-support                                0.142
 amd64                  scripts for handling many ACPI events
...
...
...
```

可以看到，乱七八糟一大堆...

这个命令也能安装软件。`dpkg -i packge_name.deb`

`apt-get install` 安装软件

也很简单啊，只要把你需要安装的软件告诉他，然后他就会按照依赖关系把需要的包一个一个都给装好：

```
$ sudo apt-get install apache2
[sudo] password for ramesh:
The following NEW packages will be installed:
apache2 apache2-mpm-worker apache2-utils
apache2.2-common libapr1 libaprutil1 libpq5
0 upgraded, 7 newly installed, 0 to remove and 26 not upgraded.
```

`apt-get remove` 卸载软件

```
$ sudo apt-get purge apache2
(or)
$ sudo apt-get remove apache2
```

两者都可以卸载, 区别是什么? 第一个是彻底卸载, 包括各种配置文件,可以理解为 `apt-get remove apache2*` 第二个只是移除二进制文件, 单纯的把apache2卸载.

除此之外, 还有 `apt-get update` , `apt-get upgrade` , `apt-get dist-upgrade` 等一系列命令... 想知道他们都是什么吗? 自己去搜吧 :)

相当的easy~

从源码安装

有时候软件仓库里并没有我们需要的软件, 或者软件仓库里的软件太老了, 而你的强迫症则迫使你安装最新的软件, 怎么办? 从源码编译安装呗!

你也许会问, 为什么是源码从新编译呢? 因为不同的系统所拥有的文件是不一样的, 在我的电脑上可以正常安装的软件, 在你那里就不行, 为什么? 缺文件啊! 所以通过源码安装的时候就会检查这些依赖库, 如果没有, 还需要你再把那些用到的依赖库给安装上. 而且, 源码比起生成的二进制文件小多了~ 还节省带宽呢!

你有会问, 为什么Windows直接点开exe就能安装了? 因为exe都是自成一派的, 程序所运行的文件都在这个exe文件当中了, 把exe里面的东西提取出来, 再到注册表里一注册, 程序就能跑, 每个软件跟其他的软件都没有半毛钱的关系, 也就不需要检查什么依赖库了(当然, 你得把*.Net*那些东西给装上).所以每个exe都比较大...而且是越来越大...

接下来就步入正题了, 怎样从源码安装?

当我们下载好程序的源码时, 第一步, 当然是把他解压了:

```
tar xvfj application.tar.bz2
```

还记得 `bz2` 吗?

然后进入这个文件:

```
cd application/
```

然后:

```
./configure
```

这是做什么? 慢慢看!

(这里没有贴输出, 因为每个软件编译的时候都不一样, 不过都是检查依赖包之类的.)

最后呢:

```
make
```

如果make成功了, 你会在当前目录下看到生成的二进制文件, 如果可以运行呢,那就可以安装到本系统上了:

```
make install
```

(也就是把那些二进制文件复制到系统中, 这个操作需要root权限).

在每个下载的规范的源代码中, 都会有一个README文件, 这个是干什么呢? 是告诉你这源代码怎么用的 :P

第十章 - LAMP 套装

这一章介绍了LAMP的安装和配置.

- **L** Linux
- **A** Apache2
- **M** Mysql
- **P** PHP

LAMP - 维基百科

安装Apache2

这一篇介绍了怎样安装带有SSL模块的Apache2.

(然后作者说了一大段apache2的优点，其实我个人还是比较倾向于nginx的，轻量级，配置简单，而且高并发.)

下载 Apache

从 apache官网 下载apache的安装包，目前的版本是 `2.4.18` (2015-12-14发行)

```
➤ wget "http://mirrors.hust.edu.cn/apache//httpd/httpd-2.4.18.tar.bz2"
--2016-01-13 14:42:33--  http://mirrors.hust.edu.cn/apache//httpd/httpd-2.4.18.tar.bz2
Resolving mirrors.hust.edu.cn (mirrors.hust.edu.cn)... 202.114.18.160
Connecting to mirrors.hust.edu.cn (mirrors.hust.edu.cn)|202.114.18.160|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5181291 (4.9M) [application/octet-stream]
Saving to: ‘httpd-2.4.18.tar.bz2’

httpd-2.4.18.tar.bz 100%[=====================>]   4.94M  3.97MB/s   in 1.2s

2016-01-13 14:42:34 (3.97 MB/s) - ‘httpd-2.4.18.tar.bz2’ saved [5181291/5181291]

➤ tar -jxf httpd-2.4.18.tar.bz2
```

安装SSL模块

```
➤ cd httpd-2.4.18/
➤ ./configure --help
`configure' configures this package to adapt to many kinds of sy
stems.

Usage: ./configure [OPTION]... [VAR=VALUE]...

To assign environment variables (e.g., CC, CFLAGS...), specify t
hem as
VAR=VALUE.  See below for descriptions of some of the useful var
iables.

Defaults for the options are specified in brackets.

Configuration:
  -h, --help              display this help and exit
      --help=short        display options specific to this packa
ge
      --help=recursive    display the short help of all the incl
uded packages
  -V, --version           display version information and exit
...
...
...
```

配置的时候有好多选项, 这里我们要安装SSL支持, 所以:

```
./configure --enable-ssl --enable-so
make
make install
```

这样就安装好了.

在 `httpd.conf` 中开启SSL

Apache的配置文件保存在 `/usr/local/apache2/conf` 目录中,(如果是apt-get安装的话, 目录则在 `/etc/apache2/conf`).

把配置文件中的 `#Include conf/extra/httpd-ssl.conf` 前面的注释符去掉保存即可.

`/usr/local/apache2/conf/extra/httpd-ssl.conf` 这个文件里面保存的就是ssl的配置, 包括公钥私钥的存放位置:

```
# egrep 'server.crt|server.key' httpd-ssl.conf
SSLCertificateFile "/usr/local/apache2/conf/server.crt"
SSLCertificateKeyFile "/usr/local/apache2/conf/server.key"
```

我们还需要创建一对公钥私钥才能让apache2正常运行, 所以:

创建公私钥

```
openssl genrsa -des3 -out server.key 2048
```

上面的命令创建了一个2048位的密钥, 其中有一步是需要你输入一个4-1023位长的密码, 记住这个密码, 以后要用到(以后也可以去掉密码的).

下一步就是创建一个 certificate request file (创建证书所用到的文件), 用到上面创建的密钥:

```
openssl req -new -key server.key -out server.csr
```

最后就是创建一个自己签发的证书了:

```
openssl x509 -req -days 365 -in server.csr -signkey server.key -
out server.crt
```

证书的时长是365天.

把证书复制过去

接着上面的步骤, 把创建的证书和密钥都放到apache的配置目录中:

```
cp server.key /usr/local/apache2/conf/
cp server.crt /usr/local/apache2/conf/
```

开启 Apache

```
/usr/local/apache2/bin/apachectl start
```

过程中需要输入刚才记录的密码:

```
Apache/2.2.17 mod_ssl/2.2.17 (Pass Phrase Dialog)
Server www.example.com:443 (RSA)
Enter pass phrase:

OK: Pass Phrase Dialog successful.
```

上面说过这个密码可以去除，这样就不需要每次开启apache2的时候都输入密码了，具体怎样做呢？谷歌会告诉你.

扩展阅读

How To Generate SSL Key, CSR and Self Signed Certificate For Apache

安装PHP

所有的Linux发行版都有php, 你可以很简单的从软件仓库安装. 但是作者还是非常建议你下载最新的PHP源代码, 然后手动编译和安装. 为什么呢? 因为这样可以很好的升级PHP版本以及打各种补丁. 这一篇介绍了如何在Linux上从源码安装PHP.

前提需要

作者在这里要求事先装好Apache2和MySQL, 但是我觉着这里没啥必要, 你也可以装Nginx啊, 也可以不需要MySQL啊, 所以你只要有一个可以运行PHP的容器即可.

即使没有容器, 也可以从命令行中运行PHP脚本.

下载安装PHP

从PHP官网下载最新的PHP版本.

(作者在这里举的例子是5.2.6, 现在早已超过这个版本了, 不过我现在在图书馆, 没网... 只能贴作者的代码)

```
# bzip2 -d php-5.2.6.tar.bz2
# tar xvf php-5.2.6.tar
```

(两种不同的解压方式, 依据你下载的格式采用不同的姿势.)

可以通过 `./configure --help` 来查看所有的配置选项, 最常用的选项是 `--prefix={install-dir-name}` , 从名字就可以看出, 这是用来确定安装目录的, 缺省选项是 `/usr/local/lib` 目录.

```
# cd php-5.2.6
# ./configure --help
```

开始编译:

```
# ./configure --with-apxs2=/usr/local/apache2/bin/apxsv --with-m
ysql
# make
# make install
# cp php.ini-dist /usr/local/lib/php.ini
```

配置 `httpd.conf` 文件

在 `/usr/local/apache2/conf/httpd.conf` 文件中添加如下几行:

```
<FilesMatch "\.ph(p[2-6]?|tml)$">
SetHandler application/x-httpd-php
</FilesMatch>
```

然后确认 `LoadModule php5_module modules/libphp5.so` 这一行代码在PHP安装的过程中添加到了 `httpd.conf` 文件中.

确认安装成功

重启Apache2:

```
# /usr/local/bin/apache2/apachectl restart
```

然后在 `/usr/local/apache2/htdocs` 目录下添加一个文件:

```
# echo '<?php phpinfo(); ?>' >> /usr/local/apache2/htdocs/test.p
hp
```

如果打开浏览器, 查看 `http://local-host/test.php` , 出现了phpinfo的相关内容, 那么就是配置好了.

安装过程中可能会遇到的错误:

Error 1 : configure: error: xml2-config not found:

如果再安装过程中遇到了一下错误:

```
# ./configure --with-apxs2=/usr/local/apache2/bin/apxs
--with-mysql
Configuring extensions
checking whether to enable LIBXML support... yes
checking libxml2 install dir... no
checking for xml2-config path...
configure: error: xml2-config not found. Please check your
libxml2 installation.
```

那么就需要你安装 `libxml2-devel` 和 `zlib-devel` 库:

```
# rpm -ivh /home/downloads/linux-iso/libxml2-devel-2.6.26-
2.1.2.0.1.i386.rpm /home/downloads/linux-iso/zlib-devel-
1.2.3-3.i386.rpm
Preparing...########################################### [100%]
1:zlib-devel########################################### [ 50%]
2:libxml2-devel######################################## [100%]
```

下载这些库并且安装上就好了.

Error 2 : configure: error: Cannot find MySQL header files.

如果你遇到了以下的错误:

```
# ./configure --with-apxs2=/usr/local/apache2/bin/apxs
--with-mysql
checking for MySQL UNIX socket location...
/var/lib/mysql/mysql.sock
configure: error: Cannot find MySQL header files under
yes. Note that the MySQL client library is not bundled
anymore!
```

则说明你没有安装MySQL, 安上就好了:

```
# rpm -ivh /home/downloads/MySQL-devel-community-5.1.25-
0.rhel5.i386.rpm
Preparing...###################################### [100%]
1:MySQL-devel-community########################### [100%]
```

安装MySQL

Ubuntu 用户直接 `sudo apt-get install mysql-server` 就好了, 其中需要输入root的密码, 记住它.

这一篇里面作者大部分都是在充数...

并没有什么技巧可言...

不过我知道一个技巧:

每次登陆mysql的时候, 是不是都要输入 `mysql -u root -p` , 然后再输入那繁杂的密码, 其实有两种方法可以解决这个问题:

第一个就是利用 `alias` -> `alias mysql='mysql -uroot -pmypassword123'` 缺省的host是本机, 而且 `-u` 和 `root` 之间可以省略空格, `-p` 和你的密码之间则不允许有空格.

第二个是利用 `~/.my.cnf` :

```
➤ cat ~/.my.cnf
[client]
user='root'
password='99b460e85139819e2f'
database='test'
```

这样当你输入 `mysql` 的时候就会直接登陆了.

想知道更多? 请搜索 `my.cnf` .

安装LAMP包

上面刚讲了作者建议从源码安装这几样东西, 这里又说了怎样用软件管理器来安装了.(不冲突, 但是真的让人感觉他是在充数啊!!!)

Yum

```
yum install httpd #安装apache2

yum install mysql-server

yum install php

yum install php-mysql
```

Apt-get

```
apt-get install nginx #安装nginx

apt-get install mysql-server #安装mysql

apt-get install php5-fpm #安装php

apt-get install php5-mysql #安装php的mysql模块
```

安装其实并不复杂, 复杂的是配置. 各种优化, 各种参数, 各种变量.

安装XAMPP

XAMPP是一个包含了Apache2, php, mysql, perl的集合包.

(类似于网上的LAMP脚本.)

从这里下载 http://sourceforge.net/projects/xampp/

我就直接贴作者的代码了, 其实没啥意思, 看看就好了:

```
# cd /opt
# tar xvzf xampp-linux-1.7.3a.tar.gz
```

开启:

```
# /opt/lampp/lampp start
```

关闭:

```
# /opt/lampp/lampp stop
```

开启特定的XAMPP服务:

```
# /opt/lampp/lampp startapache
XAMPP: Starting Apache with SSL (and PHP5)...
Note: Use startmysql in the argument to start only mysql
```

关闭:

```
# /opt/lampp/lampp stopmysql
XAMPP: Stopping MySQL...
```

这里的 `startapache` 和 `stopmysql` 都是指特定的服务.

Apache安全设置

安装配置完Apache后，进行一些安全配置还是很必要的.

黑客无所不能.

以特定的用户运行Apache

默认情况下，Apache会以 `nobody` 或者 `daemon` 运行，不过最好还是让apahce以一个没有权限的用户运行，比如: apache.

创建apache用户和用户组:

```
groupadd apache
useradd -d /usr/local/apache2/htdocs -g apache -s /bin/false apa
che
```

修改 `httpd.conf` 文件，将运行apache的用户改为apache.

```
# vi httpd.conf
User apache
Group apache
```

限制访问根目录

在 `httpd.conf` 配置如下代码以限制访问根目录:

```
<Directory />
    Options None
    Order deny,allow
    Deny from all
</Directory>
```

给 `conf` 和 `bin` 目录设置适当的权限

`conf` 和 `bin` 目录只能允许认证过的用户访问. 创建一个用户组, 把那些允许访问和修改这两个目录的用户都添加进去是一个好主意.

```
groupadd apacheadmin #创建admin用户组

chown -R root:apacheadmin /usr/local/apache2/bin #设置目录权限
chmod -R 770 /usr/local/apache2/bin

$ vi /etc/group #给这个用户组添加特定的用户
apacheadmin:x:1121:ramesh,john
```

禁止目录浏览

如果不禁止目录浏览的话, 用户有可能看到你主目录或子目录下的文件列表.(现在默认都是不可浏览的了, 不过了解一下并不是坏事 :))

```
<Directory />
    Options None
    Order allow,deny
    Allow from all
</Directory>

# (or) #

<Directory />
    Options -Indexes
    Order allow,deny
    Allow from all
</Directory>
```

禁止 `.htaccess` 文件

这个文件对于apache来说比较重要, 一些重写规则以及配置都写在这里面, 如果允许用户在子目录下重写这个文件, 那么原有默认的 `.htaccess` 文件就不管用了. 这是我们不想看到的, 所以做这个限制很有必要.

```
<Directory />
    Options None
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>
```

扩展阅读

10 Tips to Secure Your Apache Web Server on UNIX / Linux

Apachectl 和 Httpd 技巧

安装完Apache2之后，如果你想用 `apachectl` 和 `httpd` 来发挥他们最大的潜能，那你就不能仅仅使用 `start` ， `stop` ， `status` 了. 下面的九个例子向你介绍了如何高效的使用它们.

给apachectl设置一个不同的 `httpd.conf` 文件

一般来说，当你需要设置一个不同的apache2指令时，你需要修改原来的 `httpd.conf` 文件，但是利用下面的方法，你就可以新建一个，而不是修改原来的，以达到测试的目的.

```
apachectl -f conf/httpd.conf.debug
# or
httpd -k start -f conf/httpd.conf.debug
```

看一下进程:

```
ps -ef | grep http
root 25080 1 0 23:26 00:00:00 /usr/sbin/httpd -f conf/httpd.conf
.debug
apache 25099 25080 0 23:28 00:00:00 /usr/sbin/httpd -f conf/http
d.conf.debug
```

如果这个 `conf/httpd.conf.debug` 文件没有问题的话，你就可以把它复制为 `conf/httpd.conf` ，然后重启apache:

```
# cp httpd.conf.debug httpd.conf
# apachectl stop
# apachectl start
```

不修改 `httpd.conf` 文件就更换网页根目录

用 `-c` 选项可以很简单的更改网页的根目录，而不必修改配置文件.

```
# httpd -k start -c "DocumentRoot /var/www/html_debug/"
```

提升日志记录的等级

用 `-e` 选项可以更改apache2记录的日志等级

```
# httpd -k start -e debug
[Sun Aug 17 13:53:06 2008] [debug] mod_so.c(246): loaded module
auth_basic_module
[Sun Aug 17 13:53:06 2008] [debug] mod_so.c(246): loaded module
auth_digest_module
```

可用的等级有: `debug, info, notice, warn, error, crit, alert, emerg`.

显示apache已编译的模块

```
# httpd -l
Compiled in modules:
core.c
prefork.c
http_core.c
mod_so.c
```

显示apache加载的模块(动态/静态)

上一个 `-l` 选项只显示了静态编译过的模块, 这个 `-M` 则会显示动态模块和共享模块.

```
# httpd -M
Loaded Modules:
core_module (static)
mpm_prefork_module (static)
http_module (static)
so_module (static)
auth_basic_module (shared)
auth_digest_module (shared)
authn_file_module (shared)
authn_alias_module (shared)
Syntax OK
```

显示 `httpd.conf` 内所有可用的指令

这个 `-L` 的选项类似于httpd的帮助扩展, 他会显示httpd.conf中可用的变量及其参数,

```
# httpd -L
HostnameLookups (core.c)
"on" to enable, "off" to disable reverse DNS lookups, or
"double" to enable double-reverse DNS lookups
Allowed in *.conf anywhere
ServerLimit (prefork.c)
Maximum value of MaxClients for this run of Apache
Allowed in *.conf only outside <Directory>, <Files> or
<Location>
KeepAlive (http_core.c)
Whether persistent connections should be On or Off
Allowed in *.conf only outside <Directory>, <Files> or
<Location>
LoadModule (mod_so.c)
a module name and the name of a shared object file to load
it from
Allowed in *.conf only outside <Directory>, <Files> or
<Location>
```

检测修改过的 `httpd.conf` 是否有错误

当我们新更改过apache的配置文件后，我们应该先检测一下里面是否有错误的语法或者配置不正确的参数，用 `-t` 这个参数：

```
# httpd -t -f conf/httpd.conf.debug
httpd: Syntax error on line 148 of
/etc/httpd/conf/httpd.conf.debug:
Cannot load /etc/httpd/modules/mod_auth_basicso into
server:
/etc/httpd/modules/mod_auth_basicso: cannot open shared
object file: No such file or directory
Once you fix the issue, it will display Syntax OK.

# httpd -t -f conf/httpd.conf.debug
Syntax OK
```

显示 `httpd` 编译参数

```
# httpd -V
Server version: Apache/2.2.9 (Unix)
Server built:
Jul 14 2008 15:36:56
Server's Module Magic Number: 20051115:15
Server loaded:
APR 1.2.12, APR-Util 1.2.12
Compiled using: APR 1.2.12, APR-Util 1.2.12
Architecture: 32-bit
Server MPM: Prefork
threaded:
forked:
no
yes (variable process count)
Server compiled with....
-D APACHE_MPM_DIR="server/mpm/prefork"
-D APR_HAS_SENDFILE
-D HTTPD_ROOT="/etc/httpd"
-D SUEXEC_BIN="/usr/sbin/suexec"
-D DEFAULT_PIDLOG="logs/httpd.pid"
-D DEFAULT_SCOREBOARD="logs/apache_runtime_status"
-D DEFAULT_LOCKFILE="logs/accept.lock"
-D DEFAULT_ERRORLOG="logs/error_log"
-D AP_TYPES_CONFIG_FILE="conf/mime.types"
-D SERVER_CONFIG_FILE="conf/httpd.conf"
... ...
... ...
```

或者用 `-v` 显示很少的信息:

```
# httpd -v
Server version: Apache/2.2.9 (Unix)
Server built:
Jul 14 2008 15:36:56
```

按需加载特定的模块

有时候你并不需要加载全部的模块，你只需要加载某个特定的模块，怎么做呢？

还是修改 `httpd.conf` 文件:

```
<IfDefine load-ldap>
    LoadModule ldap_module modules/mod_ldap.so
    LoadModule authnz_ldap_module modules/mod_authnz_ldap.so
</IfDefine>
```

当你测试ldap的时候你会想加载所有与ldap有关的模块，所以:

```
# httpd -k start -e debug -Dload-ldap -f
/etc/httpd/conf/httpd.conf.debug
[Sun Aug 17 14:14:58 2008] [debug] mod_so.c(246): loaded
module ldap_module
[Sun Aug 17 14:14:58 2008] [debug] mod_so.c(246): loaded
module authnz_ldap_module
[Note: Pass -Dload-ldap, to load the ldap modules into
Apache]
```

注意 `-D` 参数哦~

```
# apachectl start
[Note: Start the Apache normally, if you don't want to
load the ldap modules.]
```

Apache虚拟主机

其实这里也没什么，就是增加了一个主机而已...如果是用nginx的话，相当简单的...

我就一步一步翻译好了，如果觉着简单就跳过这一节吧 :)

1. 取消 `httpd-vhosts.conf` 的注释

```
# vi /usr/local/apache2/conf/httpd.conf
Include conf/extra/httpd-vhosts.conf
```

2. 配置虚拟主机

```
1. `NameVirtualHost *:80` 声明端口号
2. `<VirtualHost *:80> </VirtualHost>` 监听80端口，里面的是主机配置
选项.
3. 下面的例子写了两个虚拟主机，照猫画虎.
```

```
# vi /usr/local/apache2/conf/extra/httpd-vhosts.conf
NameVirtualHost *:80
<VirtualHost *:80>
    ServerAdmin ramesh@thegeekstuff.com
    DocumentRoot "/usr/local/apache2/docs/thegeekstuff"
    ServerName thegeekstuff.com
    ServerAlias www.thegeekstuff.com
    ErrorLog "logs/thegeekstuff/error_log"
    CustomLog "logs/thegeekstuff/access_log" common
</VirtualHost>

<VirtualHost *:80>
    ServerAdmin ramesh@top5freeware.com
    DocumentRoot "/usr/local/apache2/docs/top5freeware"
    ServerName top5freeware.com
    ServerAlias www.top5freeware.com
    ErrorLog "logs/top5freeware/error_log"
    CustomLog "logs/top5freeware/access_log" common
</VirtualHost>
```

(作者在这里打了一手好广告 :D)

3. 检测配置文件是否有语法错误

```
# /usr/local/apache2/bin/httpd -S
VirtualHost configuration:
Syntax OK
```

上面的是配置正确的，如果出错了就会报错:

```
# /usr/local/apache2/bin/httpd -S
Warning: DocumentRoot
[/usr/local/apache2/docs/top5freeware] does not exist
Warning: ErrorLog [/usr/local/apache2/logs/thegeekstuff]
does not exist
Syntax OK
```

4. 重启测试

```
# /usr/local/apache2/bin/apachectl restart
```

apache2是通过HTTP的Host来区分不通的主机的. 你可以在本地的 `/etc/hosts` 文件中写好要测试的网址和对应的ip(127.0.0.1), 然后浏览器访问不通的网址, 虽然都是同一个ip地址(127.0.0.1), 但是却是不同的网站.

循环转存日志

其实我也不知道怎么翻译才好... 英文叫做 rotate log, 大概的意思是按照时间或大小将日志循环, 如果到了设定的日期或者日志的大小达到阀值之后, 就会自动将日志保存到别处, 然后开启一个新的日志, 这样就不会使得日志过于庞大. 如果按照时间循环的话, 就便于日志的整理和归档.

不仅是apache, 好多服务都可以这样弄, 配置文件都在 `/etc/logrotate.d/` 目录下:

```
➤ ls /etc/logrotate.d/
apport       dpkg          php5-fpm  rsyslog             unattend
ed-upgrades
apt          mysql-server  pm-utils  speech-dispatcher   upstart
cups-daemon  nginx         ppp       ufw
➤
```

作者在这里介绍了apache的配置:

```
# vi /etc/logrotate.d/apache
/usr/local/apache2/logs/access_log
/usr/local/apache2/logs/error_log {
    size 100M
    compress
    dateext
    maxage 30
    postrotate
      /usr/bin/killall -HUP httpd
      ls -ltr /usr/local/apache2/logs | mail -s
"$HOSTNAME: Apache restarted and log files rotated"
ramesh@thegeekstuff.com
    endscript
}
```

说明:

- `size 100M` 当日志文件到了100M, 系统就会转存日志. 100M可以改成100K,

100G.

- `compress` 压缩转存的日志文件
- `dateext` 给转存过的日志文件名添加日期
- `maxage` 说明转存的日志文件最多保存多久
- `postrotate and endscript` 在这两个参数之间的命令将会在转存完毕后执行. 这里是把日志通过邮件发送了出去.

添加完了可以测试下效果:

```
# /etc/cron.daily/logrotate
```

然后:

```
# ls /usr/local/apache2/logs
access_log
error_log
access_log-20110716.gz
error_log-20110716.gz
```

第十一章 - Bash脚本

~~翻译的进度越来越快了，我看今天就能把这本书搞完了~ 开心~~~

这一章介绍了几个关于bash脚本的知识和技巧.

关于.bash_*的执行顺序

这一篇介绍了下面这些文件的执行顺序:

- `/etc/profile`
- `~/.bash_profile`
- `~/.bashrc`
- `~/.bash_login`
- `~/.profile`
- `~/.bash_logout`

交互式shell的执行顺序

```
execute /etc/profile
IF ~/.bash_profile exists THEN
    execute ~/.bash_profile
ELSE
    IF ~/.bash_login exist THEN
        execute ~/.bash_login
    ELSE
        IF ~/.profile exist THEN
            execute ~/.profile
        END IF
    END IF
END IF
```

登出时:

```
IF ~/.bash_logout exists THEN
    execute ~/.bash_logout
END IF
```

需要注意的是 `/etc/bashrc` 是由 `~/.bashrc` 执行的:

```
# cat ~/.bashrc
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi
```

non-login shell的顺序

科普 nologin shell

> 当我们需要一个不允许登陆但是允许使用系统资源的用户时，就用到了nologin shell（`/usr/sbin/nologin`），比如在 `/etc/passwd` 中的各种系统账户，当登陆的shell为nologin shell 时，会提示你此账户不允许登陆，而 `/bin/false` 则是连登陆都不让登陆，只会返回False。 具体请看 :whats-the-difference-between-sbin-nologin-and-bin-false

```
IF ~/.bashrc exists THEN
    execute ~/.bashrc
END IF
```

测试.

作者在这里做了个小测试, 就是在不同的文件下写了一个 `PS1` , 看看哪个被覆盖, 其实这样做很啰嗦的, 不如在每个文件下输出一点特殊的东西, 输出什么呢? 就输出自己的文件名, 在 `~/.bashrc` 里面就加一句" `echo .bashrc >> ~/order` ", 在 `~/.bash_login` 里面就加一句" `echo bash_login >> ~/order` ", 以此类推, 最后只要看看 `~/order` 里面是什么就好了.

好麻烦的... 你感兴趣的话自己做吧, 我就不演示了哈~

For循环

一般的for循环都是这种形式的:

```
for i in {1..9}; do
    echo $i
done
```

但是还有一种形式, 也许你不知道:

```
for (( expr1; expr2; expr3 ))
do
    commands
done
```

这就是类似C语言的格式.

上面输出1-9的例子就可以写成:

```
for (( i=1; 1 < 10; i++ ))
do
    echo $i
done
```

甚至允许空条件:

```
#!/bin/bash
i=1
for (( ; ; ))
do
    sleep $i
    echo "Number: $((i++))"
done
```

```
$ ./for11.sh
Number: 1
Number: 2
Number: 3
```

多重操作:

```
#!/bin/bash
for ((i=1, j=10; i <= 5 ; i++, j=j+5))
do
echo "Number $i: $j"
done
```

```
$ ./for12.sh
Number 1: 10
Number 2: 15
Number 3: 20
Number 4: 25
Number 5: 30
```

Shell 调试

没错, shell也可以调试.

看这样一个脚本:

```
$ cat filesize.sh
#!/bin/bash
for filesize in $(ls -l . | grep "^-" | awk '{print $5}')
do
let totalsize=$totalsize+$filesize
done
echo "Total file size in current directory: $totalsize"
```

运行结果如下:

```
$ ./filesize.sh
Total file size in current directory: 652
```

添加调试信息:

```
$ cat filesize.sh
#!/bin/bash
set -xv   ## 注意这里!
for filesize in $(ls -l . | grep "^-" | awk '{print $5}')
do
let totalsize=$totalsize+$filesize
done
echo "Total file size in current directory: $totalsize"
```

输出结果如下:

```
$ ./fs.sh
++ ls -l .
++ grep '^-'
++ awk '{print $5}'
+ for filesize in '$(ls -l . | grep "^-" | awk '\''{print
$5}'\'')'
+ let totalsize=+178
+ for filesize in '$(ls -l . | grep "^-" | awk '\''{print
$5}'\'')'
+ let totalsize=178+285
+ for filesize in '$(ls -l . | grep "^-" | awk '\''{print
$5}'\'')'
+ let totalsize=463+189
+ echo 'Total file size in current directory: 652'
Total file size in current directory: 652
```

每次执行一条命令都会输出对应的命令和结果.

除了上面的方法, 还可以这样调试:

```
$ bash -xv filesize.sh
```

直接在运行的时候调试.

很有用的我会乱说~

引号

引号的作用是什么呢?

看这样一个例子:

```
➤ echo hello world!
hello world!
➤ echo "hello world!"
hello world!
➤
```

看起来没什么区别是吧?

那这样呢?

```
➤ echo hello ; world!
hello
world!: command not found
➤ echo "hello ; world!"
hello ; world!
➤
```

中间加了一个特殊字符 `;` 就报错了, 但是在添加了引号之后又成功执行了, 为什么?

引号的作用之一是确定参数.

再上面的例子中, 我们用分号隔开了 `hello world!` , 导致 `echo` 只知道hello是他的参数, 而不管 world! 了. 但是我们用引号引起来之后就取消了分号的效果. 把 `hello ; world!` 作为一个整体的参数传递给 `echo` .

单引号和双引号

跟大多数编程语言一样, 单引号里面的变量不予解析扩展, 双引号扩展变量:

```
➤ i=888
➤ echo $i
888
➤ echo "$i"
888
➤ echo '$i'
$i
➤
```

看出区别了么? 单引号里面是什么, 输出就是什么.而双引号则把变量的值扩展了.

(也许是这些我都知道了, 所以我觉着这里讲的内容都比较肤浅... 各位看官自便~)

在 Shell 脚本中读文件内容

这个名字好长... 其实也没什么东西, 无非是重定向和read的组合拳. 不过嘛, 这一拳还是很给力的 :)

假设有这样一个文件:

```
$ cat employees.txt
Emma Thomas:100:Marketing
Alex Jason:200:Sales
Madison Randy:300:Product Development
Sanjay Gupta:400:Support
Nisha Singh:500:Sales
```

还记得这个文件吗 ~

然后我们写一个脚本:

```
$ vi read-employees.sh
#!/bin/bash
IFS=:
echo "Employee Names:"
echo "---------------"
while read name empid dept
do
echo "$name is part of $dept department"
done < ~/employees.txt
```

然后我们执行一下:

```
$ chmod u+x read-employees.sh

$ ./read-employees.sh
Employee Names:
---------------
Emma Thomas is part of Marketing department
Alex Jason is part of Sales department
Madison Randy is part of Product Development department
Sanjay Gupta is part of Support department
Nisha Singh is part of Sales department
```

好玩吗?

解释一下:

`IFS` 表示的是分隔符, 因为我们给的文件里面就是用这个冒号来分隔的. 然后重定向将txt文件里面的内容都给了read, 每次读一行, 一行读三个. 然后echo再把read读到的内容输出出来.

事情就是这么个事情,情况就是这么个情况.

第十二章 - 系统性能监控

这一章介绍了十几个关于系统监控的命令.

~~最后一章啦!果真在图书馆效率就是高~(其实是因为没WIFI...)~~

Free 命令

`free` 命令显示系统的内存和交换空间的使用状态.

语法:

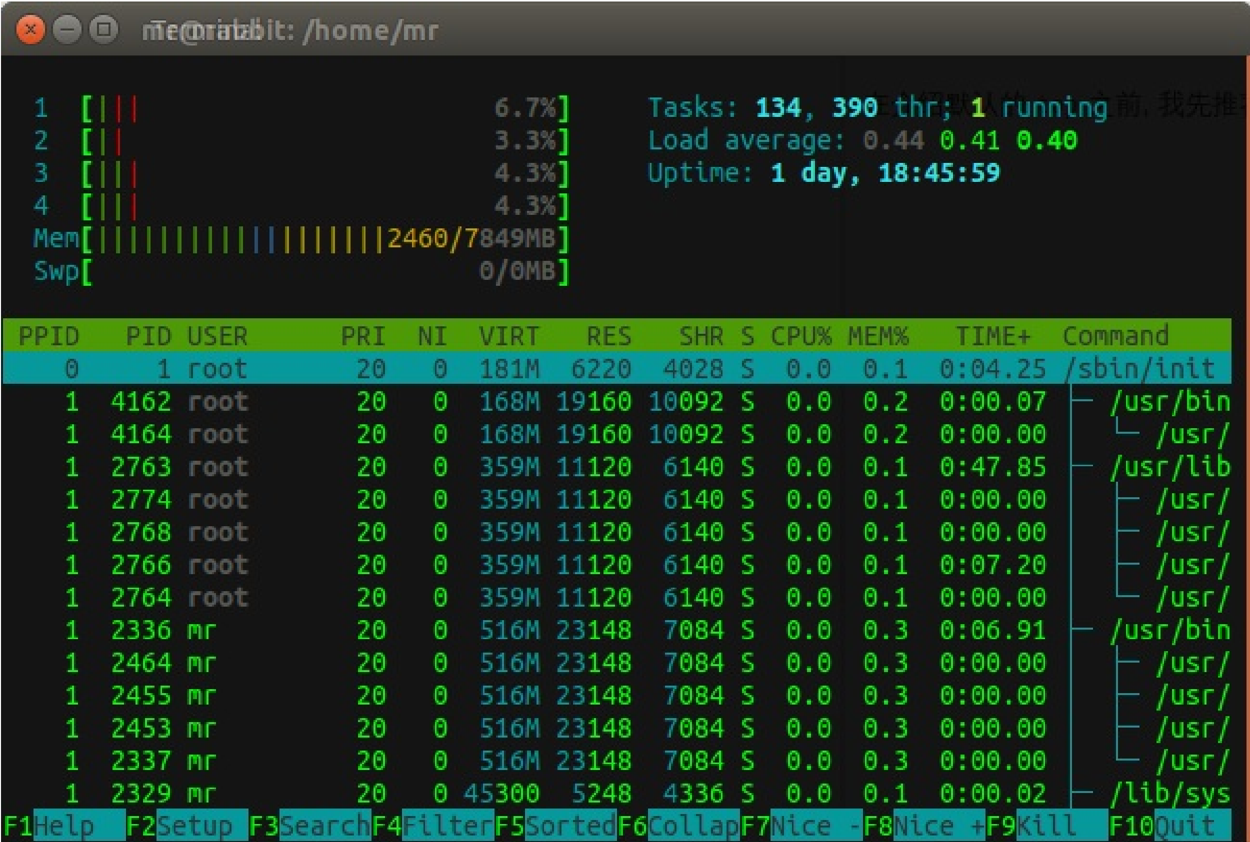
```
free [options]
```

几个参数:

- `-m` 以MB为单位显示
- `-g` 以GB为单位显示
- `-h` 以易读的方式显示
- `-t` 显示统计行

Top 命令

在介绍默认的 `top` 之前，我先推荐另一款软件 -- `htop` .



足够酷炫了吧!

开始正题 -- `top`

`top` 实时显示当前CPU运行状态，内存使用状态，系统负载状态，进程列表等.

显示的东西有点多(乱):

```
top - 18:39:29 up 1 day, 18:51,  2 users,  load average: 0.26, 0
.37, 0.40
Tasks: 241 total,   1 running, 240 sleeping,   0 stopped,   0 zo
mbie
%Cpu(s):  2.3 us,  1.5 sy,  0.0 ni, 96.0 id,  0.2 wa,  0.0 hi,
0.0 si,  0.0 st
KiB Mem:   8038040 total,  6211228 used,  1826812 free,   356876
 buffers
KiB Swap:        0 total,        0 used,        0 free.  3331852
 cached Mem

 PID  USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TI
ME+ COMMAND
 1099 root      20   0  286560   9000   6432 S   0.0  0.1   0:01
.21 polkitd
 1137 mysql     20   0    4476   1692   1536 S   0.0  0.0   0:00
.00 mysqld_safe
 1146 kernoops  20   0   45276   2680   2304 S   0.0  0.0   0:00
.71 kerneloops
```

截取了完整输出的一小段.

第一行, 显示我们的系统开机了多长时间, 几个用户登陆了, 系统负载是多少.

第二行, 显示当前有多少个进程, 有几个正在运行, 几个休眠, 几个停止, 以及几个变成了僵尸进程.

第三行, 显示了主机的CPU状态, 使用, 空闲.

第四行, 显示了主机的内存状态, 第五行显示了交换空间的使用状态.

以下的那些则是系统的进程. 默认是按照CPU的使用情况来排序的.

按照内存使用来排序

在 `top` 中按下 `f` 键, 然后通过上下键选择 `%MEM` , 再按 `s` 键选择.

搞定.

显示额外的列

按下 `f` 后再用上下键移动到你想显示的字段中，按下空格即可.

显示程序的路径信息

在 `top` 中按下 `c` 键.

```
 2542 mr        20   0 1536232 336920  73396 S   8.0  4.2  45:35
.55 compiz

10713 mr        20   0 2767056 117704  32788 S   6.0  1.5   2:55
.83 C:\Program Files (x86)\Netease\CloudMusic\cloudmusic.exe

 1847 root      20   0  532248 139948 100560 S   4.7  1.7  32:37
.36 /usr/bin/X -core :0 -seat seat0 -auth /var/run/lightdm/root/
:0 -nolisten tcp vt7 -novtswitch
 2659 mr         9 -11  501300  13680  10060 S   2.7  0.2  31:16
.66 /usr/bin/pulseaudio --start --log-target=syslog
```

每一行都显示了程序的路径信息.

显示每一个CPU核心

按下 `1` (数字1).

```
top - 18:58:39 up 1 day, 19:10,  2 users,  load average: 0.31, 0
.39, 0.40
Tasks: 241 total,   1 running, 240 sleeping,   0 stopped,   0 zo
mbie
%Cpu0  :  6.6 us,  7.3 sy,  0.0 ni, 81.1 id,  5.0 wa,  0.0 hi,
0.0 si,  0.0 st
%Cpu1  :  6.3 us,  2.0 sy,  0.0 ni, 91.1 id,  0.7 wa,  0.0 hi,
0.0 si,  0.0 st
%Cpu2  :  7.0 us,  2.7 sy,  0.0 ni, 90.3 id,  0.0 wa,  0.0 hi,
0.0 si,  0.0 st
%Cpu3  :  6.0 us,  2.3 sy,  0.0 ni, 91.6 id,  0.0 wa,  0.0 hi,
0.0 si,  0.0 st
KiB Mem:   8038040 total,  6255324 used,  1782716 free,   357560
 buffers
KiB Swap:        0 total,        0 used,        0 free.  3346672
 cached Mem
```

可以看到这台主机有四个核心.

Df 命令

`df` 命令是用来查看磁盘空间的.

嗯... 作者把他放到这里想必是用来监控磁盘使用情况的...

几个例子:

```
# df -h
➤ df -h
Filesystem       Size  Used Avail Use% Mounted on
udev             3.9G     0  3.9G   0% /dev
tmpfs            785M  9.5M  776M   2% /run
/dev/dm-1        110G   76G   29G  73% /
tmpfs            3.9G   26M  3.9G   1% /dev/shm
tmpfs            5.0M  4.0K  5.0M   1% /run/lock
tmpfs            3.9G     0  3.9G   0% /sys/fs/cgroup
/dev/sda2        237M   63M  162M  28% /boot
/dev/sda1         47M  3.4M   43M   8% /boot/efi
cgmfs            100K     0  100K   0% /run/cgmanager/fs
tmpfs            785M   96K  785M   1% /run/user/1000
```

`-h` 以人性化的方式显示.

```
➤ df -Th
Filesystem     Type      Size  Used Avail Use% Mounted on
udev           devtmpfs  3.9G     0  3.9G   0% /dev
tmpfs          tmpfs     785M  9.5M  776M   2% /run
/dev/dm-1      ext4      110G   76G   29G  73% /
tmpfs          tmpfs     3.9G   26M  3.9G   1% /dev/shm
tmpfs          tmpfs     5.0M  4.0K  5.0M   1% /run/lock
tmpfs          tmpfs     3.9G     0  3.9G   0% /sys/fs/cgroup
/dev/sda2      ext2      237M   63M  162M  28% /boot
/dev/sda1      vfat       47M  3.4M   43M   8% /boot/efi
cgmfs          tmpfs     100K     0  100K   0% /run/cgmanager/fs
tmpfs          tmpfs     785M   96K  785M   1% /run/user/1000
```

`-T` 显示分区的格式.

Du 命令

`du` -- 显示目录及其内文件的大小.

这个命令也是很常用的.

```
➤ !du
du -h
204K    ./apt
4.0K    ./sysstat
3.9M    ./account
4.0K    ./samba
4.0K    ./unattended-upgrades
8.0K    ./cups
32K     ./mysql
4.0K    ./upstart
12K     ./fsck
4.0K    ./dist-upgrade
52K     ./lightdm
57M     .
➤
```

更多信息 - `man du`

Lsof 命令

`lsof`, **ls** **o**pen **f**iles.

列举当前打开的文件, 文件也包括网络连接(socket 文件),设备文件, 以及目录文件.

`lsof` 的输出包括了如下几列:

```
➤ lsof | head
COMMAND     PID   TID             USER   FD      TYPE
  DEVICE SIZE/OFF       NODE NAME
systemd       1                   root  cwd   unknown
                             /proc/1/cwd (readlink: Permission d
enied)
systemd       1                   root  rtd   unknown
                             /proc/1/root (readlink: Permission
denied)
systemd       1                   root  txt   unknown
                             /proc/1/exe (readlink: Permission d
enied)
systemd       1                   root NOFD
                             /proc/1/fd (opendir: Permission den
ied)
kthreadd      2                   root  cwd   unknown
                             /proc/2/cwd (readlink: Permission d
enied)
kthreadd      2                   root  rtd   unknown
                             /proc/2/root (readlink: Permission
denied)
kthreadd      2                   root  txt   unknown
                             /proc/2/exe (readlink: Permission d
enied)
kthreadd      2                   root NOFD
                             /proc/2/fd (opendir: Permission den
ied)
ksoftirqd     3                   root  cwd   unknown
                             /proc/3/cwd (readlink: Permission d
enied)
```

说一下这几列都代表了什么:

- `COMMAND` 进程的命令
- `PID` 进程ID
- `USER` 用户名
- `FD` 文件描述符
- `TYPE` 文件类型
- `DEVICE` 设备编号
- `SIZE` 文件大小
- `NODE` 节点编号
- `NAME` 文件的绝对路径名

显示系统打开的所有文件

```
➤ lsof | less
```

只是显示出来并没有太大的作用, 但是统计的话, 就另当别论了:

```
➤ lsof | wc -l
73610
```

可以看到我一共打开了73610个文件.

查看特定用户打开的文件

`-u` 参数:

```
# lsof –u ramesh
vi 7190 ramesh 475196 /bin/vi txt REG 8,1 474608
sshd 7163 ramesh 3u IPv6 15088263 TCP dev-db:ssh->abc-12-12-12-1
2.socal.res.rr.com:2631 (ESTABLISHED)
```

查看特定程序打开的文件

`lsof progream_name` :

```
➤ lsof /usr/bin/vi
COMMAND    PID USER   FD    TYPE DEVICE SIZE/OFF    NODE NAME
vi       12760   mr txt     REG  252,1  2837384 1051186 /usr/bin/v
im.gnome
```

其实还有一些好玩的, 还有参数介绍什么的, 我博客里有写.(没网啊, 不能贴具体的链接... 麻烦各位看官自己找找去... 里面还有个类似的软件 [fuser], 很是强大 http://wrfly.kfd.me)

Vmstat 命令

`vmstat` 显示了系统的内存, 交换空间, IO, 甚至是CPU的信息.

下面的例子每隔一秒显示 `vmstat` 的状态, 一共显示100次:

```
➤ vmstat 1 100
procs -----------memory---------- ---swap-- -----io---- -system--- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0      0 1580700 390520 3347592    0    0    22   152  174  339 23  9 67  1  0
 0  0      0 1580700 390520 3347592    0    0     0     0  557 10101  2  2 96  0  0
 0  0      0 1580204 390520 3347592    0    0     0    64  539 10151  2  2 96  0  0
 0  0      0 1580128 390520 3348252    0    0     0     0  603 10847  3  3 93  0  0
^C
➤  #嗯, 我给它康楚C了
```

`Procs` 部分

- `r` 可以运行的进程
- `b` 禁止运行的进程

`Memory` 部分

- `Swpd` 已经用了的交换空间大小
- `Free` 可用的内存大小
- `Buff` 用了的Buff
- `Cache` 用了的Cache

`Swap` 部分

- `Si` 每秒从磁盘写入到内存的大小

- `So` 每秒从内存写到磁盘的大小

`IO` 部分

- `Bi` 从磁盘接收的块儿
- `Bo` 发送到磁盘的块儿

`System` 部分

- `In` 每秒中断次数
- `Cs` 每秒上下文切换的次数

`CPU` 部分

- `Us` 用户代码使用CPU的时间
- `Sy` 内核代码使用CPU的时间
- `Id` 空闲时间
- `Wa` 等待IO的时间

Netstat 命令

`netstat` 命令显示网络连接的状态.

显示当前活动的网络连接

```
➤ netstat -an
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address
    State
tcp        0      0 127.0.0.1:44244         0.0.0.0:*
    LISTEN
tcp        0      0 127.0.1.1:53            0.0.0.0:*
    LISTEN
tcp        0      0 127.0.0.1:25            0.0.0.0:*
    LISTEN
tcp        0      0 127.0.0.1:3306          0.0.0.0:*
    LISTEN
... ... ...
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags       Type       State         I-Node   Path
unix  2      [ ]         DGRAM                    22370    /run/
user/1000/systemd/notify
unix  2      [ ACC ]     STREAM     LISTENING     23426    @/tmp
/.ICE-unix/2548
unix  2      [ ACC ]     STREAM     LISTENING     22371    /run/
user/1000/systemd/private
unix  2      [ ACC ]     SEQPACKET  LISTENING     8934     /run/
udev/control
unix  2      [ ACC ]     STREAM     LISTENING     20436    /run/
user/1000/keyring/control
unix  2      [ ACC ]     STREAM     LISTENING     24609    /tmp/
fcitx-socket-:0
unix  2      [ ACC ]     STREAM     LISTENING     27013    /tmp/
sogou-qimpanel-cell
```

显示运行程序的进程号/程序名

```
➤ netstat -tapn
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address
    State       PID/Program name
tcp        0      0 127.0.0.1:44244         0.0.0.0:*
    LISTEN      9720/1452662683780-
tcp        0      0 127.0.1.1:53            0.0.0.0:*
    LISTEN      -
tcp        0      0 127.0.0.1:25            0.0.0.0:*
    LISTEN      -
tcp        0      0 127.0.0.1:3306          0.0.0.0:*
    LISTEN      -
tcp        0      0 127.0.0.1:587           0.0.0.0:*
    LISTEN      -
tcp        0      0 127.0.0.1:47034         127.0.0.1:44244
    ESTABLISHED 9680/editor
tcp        0      0 127.0.0.1:44244         127.0.0.1:47034
    ESTABLISHED 9720/1452662683780-
tcp        0      0 127.0.0.1:47038         127.0.0.1:44244
    ESTABLISHED 9680/editor
tcp        0      0 127.0.0.1:44244         127.0.0.1:47038
    ESTABLISHED 9720/1452662683780-
tcp6       0      0 :::1080                 :::*
    LISTEN      31588/shadowsocks
tcp6       0      0 :::1088                 :::*
    LISTEN      31589/shadowsocks
```

在最后一列有运行的进程号/程序名.

显示路由表

```
➤ netstat --route
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Wind
ow  irtt Iface
link-local      *               255.255.0.0     U         0 0
       0 docker0
172.17.0.0      *               255.255.0.0     U         0 0
       0 docker0
```

显示RAW网络统计

```
➤ netstat --statistics --raw
Ip:
    3046724 total packets received
    20 forwarded
    0 incoming packets discarded
    3045898 incoming packets delivered
    1985453 requests sent out
    48 outgoing packets dropped
    1611 dropped because of missing route
Icmp:
    474 ICMP messages received
    0 input ICMP message failed.
    ICMP input histogram:
        destination unreachable: 471
        echo requests: 3
    1999 ICMP messages sent
    0 ICMP messages failed
    ICMP output histogram:
        destination unreachable: 1996
        echo replies: 3
IcmpMsg:
        InType3: 471
        InType8: 3
        OutType0: 3
        OutType3: 1996
UdpLite:
IpExt:
```

```
    InNoRoutes: 38
    InMcastPkts: 143
    OutMcastPkts: 164
    InBcastPkts: 310
    OutBcastPkts: 6
    InOctets: 5251249872
    OutOctets: 149575262
    InMcastOctets: 26980
    OutMcastOctets: 27820
    InBcastOctets: 32799
    OutBcastOctets: 284
    InNoECTPkts: 3750022
➤
```

(显示太占篇幅了...)

其他

- `netstat --tcp --numeric` 列出本机的TCP连接
- `netstat --tcp --listening --programs` 显示系统正在监听的端口以及程序

```
➤ netstat --tcp --listening --programs -n
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address
    State
tcp        0      0 127.0.0.1:44244         0.0.0.0:*
    LISTEN
tcp        0      0 127.0.1.1:53            0.0.0.0:*
    LISTEN
tcp        0      0 127.0.0.1:25            0.0.0.0:*
    LISTEN
tcp        0      0 127.0.0.1:3306          0.0.0.0:*
    LISTEN
tcp        0      0 127.0.0.1:587           0.0.0.0:*
    LISTEN
tcp6       0      0 :::1080                 :::*
    LISTEN
tcp6       0      0 :::1088                 :::*
    LISTEN
```

- `netstat -rnC` 显示路由缓存

Sysctl 命令

Linux系统可以用Sysctl命令随时改变内核参数而不用重启.

```
# sysctl -a
dev.cdrom.autoclose = 1
fs.quota.writes = 0
kernel.ctrl-alt-del = 0
kernel.domainname = (none)
kernel.exec-shield = 1
net.core.somaxconn = 128
net.ipv4.tcp_window_scaling = 1
net.ipv4.tcp_wmem = 4096 16384 131072
net.ipv6.route.mtu_expires = 600
sunrpc.udp_slot_table_entries = 16
vm.block_dump = 0
```

修改 `/etc/sysctl.conf` 文件, 永久改变内核参数

内核参数一般是在启动的时候加载的, 但是也可以用这个命令来修改.

```
# vi /etc/sysctl.conf
... ... ...
# sysctl -p
```

临时修改内核参数

```
sysctl -w {variable-name=value}
```

其中有点意思的还是 `net.ipv4.ip_default_ttl` 这个, 这个ttl就是当你ping本机, 或者别人ping你的时候显示的数字, 如果你改成别的了, 那就会显示别的, 比如:

```
sysctl -w net.ipv4.ip_default_ttl=233
```

这样以后, 你ping自己的时候ttl就会是233.

Nice 命令

系统内核依据进程的优先级决定了进程使用CPU的时间，而这个 `nice` 命令则是改变进程优先级的一个工具.

进程优先级的范围，是[-20, 20]，不能多也不能少.

需要 注意 的是，-20 是最高的优先级，系统会优先运行这个进程，而 20则是最低的优先级.

还需要 注意 的是，只有root才能设置负数的优先级，普通用户只能设置从0到20的优先级.

语法也很简单:

```
nice -[value from -20 to 20] command
```

举例子:

```
$ ./nice-test.sh &
[3] 13009
$ ps axl | grep nice-test
0 509 13009 12863 17 0 4652 972 wait S pts/1 0:00 /bin/bash ./nice-test.sh

[注意: 第六列的优先级的值为0.]
```

```
$ nice -10 ./nice-test.sh &
[1] 13016
```

然后

```
$ ps axl | grep nice-test
0 509 13016 12863 30 10 4236 0:00 /bin/bash ./nice-test.sh

[注意: 这里的优先级变成10了(优先级降低了)]
```

下面在看一个错误:

```
$ nice --10 ./nice-test.sh &
[1] 13021
$ nice: cannot set priority: Permission denied
```

为啥哩?

刚才我们说过, 只有root才能设置负数的优先级:

```
# nice --10 ./nice-test.sh &
[1] 13060
# ps axl | grep nice-test
4 0 13060 13024 10 -10 5388 964 wait S< pts/1 0:00 /bin/bash ./n
ice-test.sh 964 wait S< pts/1
```

Renice 命令

顾名思义, `renice` 是用来重新设置进程优先级的.

语法:

```
renice [-20到20的优先级] -p [PID]
```

降低一个进程的优先级

```
$ ps axl | grep nice-test
0 509 13245 13216 30 10 5244 968 wait SN pts/1 0:00 /bin/bash ./
nice-test.sh
```

然后:

```
$ renice 16 -p 13245
13245: old priority 10, new priority 16

$ ps axl | grep nice-test
0 509 13245 13216 36 16 5244 968 wait SN pts/1 0:00 /bin/bash ./
nice-test.sh
```

优先级变成16了, 降低了.

增加一个进程的优先级

```
$ ps axl | grep nice-test
0 509 13254 13216 30 10 4412 968 wait SN pts/1 0:00 /bin/bash ./
nice-test.sh
```

然后增加它的优先级(以普通用户的身份):

```
$ renice 5 -p 13254
renice: 13254: setpriority: Permission denied
Login as root to increase the priority of a running process
```

报错了, 说我们权限不足, 要用root才行.

```
$ su -
# renice 5 -p 13254
13254: old priority 10, new priority 5

# ps axl | grep nice-test
0 509 13254 13216 25 5 4412 968 wait SN pts/1 0:00 /bin/bash ./n
ice-test.sh
```

这样就可以了.

Kill 命令

`kill` 是用来杀进程的, 怎么杀? 发信号.

```
语法: kill [选项] [进程号]
```

通俗的说, 就是给一个进程发送各种信号, 这里讲的是杀死, 那咱们就说杀死进程.

比如, 有个脚本的PID是2333, 跑到半路跑不动了, 没反应了, 你又很心急, 这咋弄, 算了,重启吧! 当然, 可不是系统重启, 而是让这个进程重启, 重启第一步, 把他干掉.

你可以通过干掉进程号的方式把他干掉:

```
kill 2333
```

如果干不掉呢, 再这样, 终结它:

```
kill -9 2333
```

或者呢, 直接:

```
pkill my_stuipd_script.sh
```

这个pkill是kill的进化版~

Ps 命令

~~终于到100了啊, 还有点小激动呢~~~

`ps` -- process status -- 输出所有活动的进程信息

显示当前系统中所有正在运行的进程

用 `ps aux`

```
➤ ps aux | head
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME
 COMMAND
root          1  0.0  0.0 185576  6220 ?        Ss    1月12   0:0
4 /sbin/init splash
root          2  0.0  0.0      0     0 ?        S     1月12   0:0
0 [kthreadd]
root          3  0.0  0.0      0     0 ?        S     1月12   0:0
0 [ksoftirqd/0]
root          5  0.0  0.0      0     0 ?        S<    1月12   0:0
0 [kworker/0:0H]
root          7  0.0  0.0      0     0 ?        S     1月12   1:0
5 [rcu_sched]
root          8  0.0  0.0      0     0 ?        S     1月12   0:0
0 [rcu_bh]
root          9  0.0  0.0      0     0 ?        S     1月12   0:3
0 [rcuos/0]
root         10  0.0  0.0      0     0 ?        S     1月12   0:0
0 [rcuob/0]
root         11  0.0  0.0      0     0 ?        S     1月12   0:0
0 [migration/0]
```

(我截取了一小段.)

打印进程树

再加一个 `-f` 的选项:

```
➤ ps auxf | head
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME
 COMMAND
root           2  0.0  0.0      0     0 ?        S     1月12   0:0
0 [kthreadd]
root           3  0.0  0.0      0     0 ?        S     1月12   0:0
0  \_ [ksoftirqd/0]
root           5  0.0  0.0      0     0 ?        S<    1月12   0:0
0  \_ [kworker/0:0H]
root           7  0.0  0.0      0     0 ?        S     1月12   1:0
5  \_ [rcu_sched]
root           8  0.0  0.0      0     0 ?        S     1月12   0:0
0  \_ [rcu_bh]
root           9  0.0  0.0      0     0 ?        S     1月12   0:3
0  \_ [rcuos/0]
root          10  0.0  0.0      0     0 ?        S     1月12   0:0
0  \_ [rcuob/0]
root          11  0.0  0.0      0     0 ?        S     1月12   0:0
0  \_ [migration/0]
root          12  0.0  0.0      0     0 ?        S     1月12   0:0
0  \_ [watchdog/0]
```

显示指定用户的进程

再加一个 `U` 的参数:

```
➤ ps fU mr | head
  PID TTY      STAT   TIME COMMAND
 2338 ?        Ss     0:01 /sbin/upstart --user
 2419 ?        S      0:00  \_ upstart-udev-bridge --daemon --us
er
 2431 ?        Ss     2:39  \_ dbus-daemon --fork --session --ad
dress=unix:abstract=/tmp/dbus-nAkkVRUjFl
 2443 ?        Ss     0:00  \_ /usr/lib/x86_64-linux-gnu/hud/win
dow-stack-bridge
 2456 ?        Ss     0:01  \_ gpg-agent --daemon --sh
 2483 ?        Sl     0:18  \_ /usr/lib/x86_64-linux-gnu/bamf/ba
mfdaemon
```

其实 `ps` 远远不止这点东西. 还有排序啊, 筛选啊一些东西, 都比较好玩和有趣.

Sar 命令

> sar - Collect, report, or save system activity information.

`sar` 是一个非常优秀的系统监视工具.(首先要确保你安装了 `sysstat`)

其实我并没有用过这个工具, 原作中还有两个sa1和sa2, 由于我不知道他在说什么, (里面错误真的很明显), 我就给删去了...下面的这几个命令我也没有验证成功, 说是我电脑收集的数据不够...醉的不行... 各位看官权当看个笑话吧 :D

然后推荐一个系统监视工具, 还是自带绘图功能的, 更直观一点**-- `munin`**

显示CPU状态

```
# sar –u
Linux 2.6.9-42.ELsmp (dev-db) 01/01/2009
12:00:01 AM  CPU %user %nice %system %iowait %idle
12:05:01 AM 95.45 all 3.70 0.00 0.85 0.00
12:10:01 AM 94.16 all 4.59 0.00 1.19 0.06
12:15:01 AM 95.11 all 3.90 0.00 0.95 0.04
12:20:01 AM 94.93 all 4.06 0.00 1.00 0.01
12:25:01 AM 95.23 all 3.89 0.00 0.87 0.00
12:30:01 AM 95.23 all 3.89 0.00 0.87 0.00

...

Average: 94.29 all 4.56 0.00 1.00 0.15
```

显示磁盘IO状态

(卧槽真是虎头蛇尾啊... 作者在这里排的版乱的不行不行的... 复制代码都复制不上... 抱歉了各位...)

那我就不写原作说的了, 写个结束语吧

看上面的代码你也知道, 这本书都是2009年写的了, 好多东西都变了, 好多东西也没变.

陈旧的东西就让他陈旧下去吧, 无门无法阻挡他们的腐烂, 但是, 沉舟侧畔千帆过, 病树前头万木春. 倘若没有这些陈腐的东西, 又哪来的万木春呢?

我们从中汲取的是营养, 是我们曾经的未知. 是前人的思想. 就拿这个 `sar` 命令来说, 现在的自动化运维很发达了, 监控软件什么的也层出不穷, 那么还有必要学一个2009年的软件吗? 我个人觉得还是有必要的(尽管这一节没写多少东西,但, 聪明的, 你会去主动学习的,对么?), 因为这是前人的一种思维方法, 我们可以从中学习核心思想, 优化操作方式, 制造出新的轮子.

当我第一次看到这本书的时候, 我是被他前几节所介绍的气门怪招吸引了, 我相信你也是, 可是你有没有发现, 越到最后越有种黔驴技穷的感觉, 工具不再那么有趣了, 命令也不再那么简单了, 相反, 命令是越来越复杂, 参数也是越来越多, 你还有兴趣看下去吗?

我们吃饭的时候, 总是挑选那写自己喜欢的, 好吃的, 然后把不喜欢的放一边, 等到肚子饿了, 也许会想起我的饭还没吃完, 不过剩下的都是自己不喜欢的了, 你会怎么办?

我们常说做这个没用, 做那个没用, 等用到的时候呢?

~~(妈蛋我可是毕不了业的人, 哪有资格说这番话.)~~

每一条命令都有它的精髓, 都有它的灵魂, 从它被设计出来的时候就有了.

我们所要做的, 则是继承那些灵魂, 并, 发扬他们.

第十三章 - 额外的技巧

让cd对大小写不敏感

默认的 `cd` 是对大小写敏感的:

```
➤ ls
hello  Hello  hi  Hi
➤ cd h
hello/ hi/
➤ cd h

➤ cd H
Hello/ Hi/
➤ cd H
```

虽然这是Linux与Windows阵营的不同,但有时候还是不要区分大小写的好,毕竟我们有 `tab` 补全嘛.

所以:

```
bind "set completion-ignore-case on"
```

这条命令的作用就是让补全不区分大小写

```
➤ bind "set completion-ignore-case on"
➤ cd h
hello/ Hello/ hi/    Hi/
➤ cd h
```

你可以把他写到 `.bashrc` 里面,以长久生效.

如果觉得还是区分大小写比较好呢,那就再把他关闭:

```
bind "set completion-ignore-case off"
```

或者从 `.bashrc` 里面删除上面那句就好了.

SSH长连接

我们知道怎样可以在连接ssh远程服务器的时候不输入密码, 但是每次连接的时候都建立一个新的连接未免过于啰嗦, 尤其是经常时不时就登上服务器的时候, 每次连接都耗费了我们的耐心.

这个时候, 如果SSH像TCP长连接那样就好了, 一直建立着, 不会断开.

你别说, 还真有.

不过名字就不叫SSH长连接了, 英文名叫 "SSH’s ControlMaster".

也就是一个小管家, 管理着你的那些连接. 比如, 你登陆了服务器A, 当你退出服务器A的时候, 小管家并不会断开连接, 尽管你看到确实是已经与服务器断开了. 当你再次登陆服务器A的时候, 你就回发现速度明显快了许多, 而且第一次登陆时欢迎的banner也不见了, 这是为什么? 就是因为小管家一直在连接着服务器, 在你看来, 你已经与服务器断开连接了, 但是服务器却认为你一直在线. 而且你用 `ss -ant` 查看的时候也会发现连接是建立着的.

那怎么做呢?

在 `~/.ssh/config` 文件里, 添加如下几句话:

```
Host *
ControlMaster auto
ControlPath ~/.ssh/master-%r@%h:%p
```

解释一下都是什么意思:

- `Host *` 对于所有主机都让小管家托管
- `ControlMaster auto` 开启小管家的自动控制
- `ControlPath` 托管的凭据在哪儿(一切皆文件)

当然, 你也可以指定仅适用于某些主机:

```
HOST 192.168.0.*
HOST *.com
```

如此之类的.

然后, 贴一下我的配置文件:

```
TCPKeepAlive=yes
ServerAliveInterval=15
ServerAliveCountMax=6
Compression=yes
ControlMaster auto
ControlPath /tmp/%r@%h:%p
ControlPersist yes
```

刚才更新github的时候建立了一个连接:

```
➤ l git@github.com\:22
srw------- 1 mr mr 0  1月 19 18:12 git@github.com:22=
➤
```

我看可以看到, 这是一个socket文件, 并且:

```
➤ ss -ant4 | grep -E ':22'
ESTAB      0      0      10.177.46.227:40182              192.30
.252.131:22
➤
```

也是可以看到这个已经建立的连接的. 节省了以后再次建立连接的时间. 真的很快 :)

RAR命令

虽说RAR是商业软件, 但是并不妨碍我们使用它啊.

Linux下也有 `rar` 命令呢!

建立一个rar压缩包

语法:

```
rar a {.rar file-name} {file-name}
```

很简单:

```
➤ rar a sh.rar *.sh

RAR 5.30 beta 2   Copyright (c) 1993-2015 Alexander Roshal   4 A
ug 2015
Trial version             Type RAR -? for help

Evaluation copy. Please register.

Creating archive sh.rar

Adding    4.sh
      OK
Adding    cal_random.sh
      OK
Adding    cmd.sh
      OK
Adding    ifs.sh
      OK
Adding    redirect.sh
      OK
Adding    std.sh
      OK
Done
➤
```

支持正则，支持递归。

解压rar压缩包

解压也很简单：

```
➤ unrar e sh.rar

UNRAR 5.30 beta 2 freeware      Copyright (c) 1993-2015 Alexande
r Roshal


Extracting from sh.rar

Extracting  4.sh
      OK
Extracting  cal_random.sh
      OK
Extracting  cmd.sh
      OK
Extracting  ifs.sh
      OK
Extracting  redirect.sh
      OK
Extracting  std.sh
      OK
All OK
➤
```

列出压缩包内的内容

```
➤ unrar l sh.rar

UNRAR 5.30 beta 2 freeware      Copyright (c) 1993-2015 Alexande
r Roshal

Archive: sh.rar
Details: RAR 4

 Attributes      Size     Date    Time   Name
----------- ---------  ---------- -----  ----
 -rw-rw-r--        25  2016-01-18 16:10  4.sh
 -rw-rw-r--       542  2015-12-24 14:08  cal_random.sh
 -rw-rw-r--        41  2016-01-18 11:02  cmd.sh
 -rw-rw-r--        27  2016-01-17 19:19  ifs.sh
 -rw-rw-r--        50  2016-01-18 13:24  redirect.sh
 -rw-rw-r--        56  2016-01-18 11:08  std.sh
----------- ---------  ---------- -----  ----
                  741                    6

➤
```

压缩加密

```
➤ rar a -p"Hello" sh.rar *.sh

RAR 5.30 beta 2   Copyright (c) 1993-2015 Alexander Roshal   4 A
ug 2015
Trial version             Type RAR -? for help

Evaluation copy. Please register.

Creating archive sh.rar

Adding    4.sh
      OK
Adding    cal_random.sh
      OK
Adding    cmd.sh
      OK
Adding    ifs.sh
      OK
Adding    redirect.sh
      OK
Adding    std.sh
      OK
Done
➤
```

Comm 命令

`comm` 全名是 `compare`，也就是用来对比文件的.

但是这个对比是有要求的，要求就是两个文件必须先排好序，然后 `comm` 再一行一行的对比.

```
➤ cat 1
1
2
4 #
5 #
➤ cat 2
1
2
3 #
4 #
➤ comm 1 2
        1
        2
    3 #文件2的不同
        4
5 #文件1的不同
➤
```

可以结合 `diff` 命令对比一下区别:

```
➤ diff 1 2
2a3
> 3
4d4
< 5
```

下面的更直观一点:

```
➤ diff -u 1 2
--- 1    2016-01-19 19:28:28.979078030 +0800
+++ 2    2016-01-19 19:28:36.059135875 +0800
@@ -1,4 +1,4 @@
 1
 2
+3
 4
-5
➤
```

Tee 命令

有时候我们需要把命令输出的内容保存为日志, 还想着实时看到命令输出的内容, 怎么办?

> tee - read from standard input and write to standard output and files

从标准输入中读取, 输出的同时并写入到文件中。

举几个栗子:

1.输出的同时写入到文件中:

```
➤ for i in {1..9};do > $RANDOM; done #创建了9个文件
➤ ls #查看一下, 的确创建成功了.
12681  16993  20566  21822  22742  25812  31954  5965  9458
➤ ls | tee files_of_here #然后用tee将输出保存到文件中
12681
16993
20566
21822
22742
25812
31954
5965
9458
files_of_here #这里有些奇怪, 为什么ls的时候多了一个这样的文件呢?
➤ cat files_of_here
12681
16993
20566
21822
22742
25812
31954
5965
9458
files_of_here # 解答上面的问题, 因为ls和tee是同时进行的,
# 所以, tee 创建文件的时候, ls 还没读完整个目录中的文件名,
# 所以才会有一个files_of_here这样的文件, 如果还不明白, 看下面的例子.
➤
```

再一个例子:

```
➤ ls | tee 000 #我们创建一个000的文件
12681
16993
20566
21822
22742
25812
31954
5965
9458
files_of_here
➤ cat 000
12681
16993
20566
21822
22742
25812
31954
5965
9458
files_of_here
➤ ls
000  12681  16993  20566  21822  22742  25812  31954  5965  9458
  files_of_here
➤
```

看出区别了么？由于ls和tee是同时进行的，所以谁先得到系统的资源是不一定的，当ls先读取目录之后，就不会输出tee创建的文件，而如果是tee先创建了文件，ls再读取目录的话，就会显示tee创建的新文件。

2.tee和管道

本来tee就是从管道中读取的，不过，他也能够输出到管道中，起到一个数据中转的作用：

```
➤ cat resout
Give me a...
... flag!
➤ cat resout | tee content_of_resout | grep flag
... flag!
➤ cat content_of_resout
Give me a...
... flag!
➤
```

3. `tee` 和 `vim`

如果用 `vim` 打开了一个没有写权限的文件, 但是你已经写了很多东西, 那该怎么办?

`tee` 来拯救你.

在 `vim` 中命令模式下:

```
:w !sudo tee %
```

更多的解释在这里:http://stackoverflow.com/questions/2600783/how-does-the-vim-write-with-sudo-trick-work

Od 命令

`od` 的用途是以某种格式查看文件, 这里的格式, 指的是进制的格式, 比如八进制, 十进制.

看这样一个文件:

```
$ cat sample-file.txt

abc     de
f     h
```

用 `od` 查看:

```
# od -c special-chars.txt

0000000   a   b   c      \t   d   e  \n   f
 \t   h
0000020  \n
0000021

# od -bc special-chars.txt

0000000 141 142 143 040 011 144 145 012 146 040 040 040 040 040
011 150
          a   b   c      \t   d   e  \n   f
 \t   h
0000020 012
         \n
0000021
```

其中:

- `b` 显示8进制
- `c` 显示原ASCII码

(其实除了 `od` , 还有更好用的 `xxd` , 个人觉着比 `od` 好一些.)